НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки

(повне найменування інституту, факультету)

Автоматизованих систем обробки інформації і управління

(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (ініціали, прізвище)
“ ” 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення інформаційних управляючих систем та технологій»

спеціальності «121 Інженерія програмного забезпечення»

на тему *Веб-застосування для організації рейтингових турнірів з жеребкуванням та двома конкуруючими сторонами*

Виконав: студент IV курсу, групи *ІП-63 Кошовець Євгеній Павлович* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *ас., Нечай Д.О.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Рецензент:

ас. каф. ОТ Шемсединов Т.Г.
посада,науковий ступінь,вчене звання,прізвище,ім’я, по батькові (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

UNICHECK

Власник документу:
Попенко Володимир Дмитрович

Дата перевірки:
11.06.2020 00:54:11 EEST

Дата звіту:
11.06.2020 23:17:28 EEST

ID перевірки:
1003947610

Тип перевірки:
Doc vs Internet + Library

ID користувача:
77149

Назва документу: Koshovets_ip63

ID файлу: 1003962809 **Кількість сторінок:** 55 **Кількість слів:** 9038 **Кількість символів:** 68032 **Розмір файлу:** 150.16 KB

15.5% Схожість

Найбільша схожість: 7.28% з джерело бібліотеки. **ID файлу:** 1000778350

11.8% Схожість з Інтернет джерелами	184	Page 57
15.4% Текстові збіги по Бібліотеці акаунту	607	Page 62

0.32% Цитат

Цитати	1	Page 63

Вилучення переліку посилань вимкнено

0% Вилучень

Вилучений текст відсутній

Підміна символів

Заміна символів	1

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення інформаційних управляючих систем та технологій*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___” _____2020 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Кошовцю Євгенію Павловичу
(прізвище, ім’я, по батькові)

1. Тема проєкту «*Веб-застосування для організації рейтингових турнірів з жеребкуванням та двома конкуруючими сторонами*»

керівник проєкту Нечай Дмитро Олександрович , ас.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “07” травня 2020 р. №1081-с

2. Термін подання студентом проєкту «08» *червня 2020 року*

3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, засоби розробки, технічні рішення, архітектура програмного забезпечення
3) Аналіз якості програмного забезпечення та опис випробувань
4) Розгортання програмного забезпечення, керівництво користувача

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема структурна класів шару сервісів

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» *березня 2020 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*15.03.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*22.03.2020*	
3.	*Постановка та формалізація задачі*	*25.03.2020*	
4.	*Аналіз вимог до програмного забезпечення*	*29.03.2020*	
5.	*Алгоритмізація задачі*	*05.04.2020*	
6.	*Моделювання програмного забезпечення*	*12.04.2020*	
7.	*Обґрунтування використовуваних технічних засобів*	*15.04.2020*	
8.	*Розробка архітектури програмного забезпечення*	*20.04.2020*	
9.	*Розробка програмного забезпечення*	*30.04.2020*	
10.	*Налагодження програми*	*10.05.2020*	
11.	*Виконання графічних документів*	*17.05.2020*	
12.	*Оформлення пояснювальної записки*	*24.05.2020*	
13.	*Подання ДП на попередній захист*	*28.05.2020*	
14.	*Подання ДП рецензенту*		
15.	*Подання ДП на основний захист*	*12.06.2020*	

Студент _____ Євгеній КОШОВЕЦЬ
(підпис)

Керівник _____ Дмитро НЕЧАЙ
(підпис)

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-6316.045440.01.81	Пояснювальна записка	59	
3	А4	КПІ.ІП-6316.045440.02.91	Технічне завдання	11	
4	А4	КПІ.ІП-6316.045440.03.13	Опис програми	40	
5	А4	КПІ.ІП-6316.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-6316.045440.05.34	Керівництво користувача	9	
7	А3	КПІ.ІП-6316.045440.06.99.СС	Схема структурна варіантів використання	1	
8	А3	КПІ.ІП-6316.045440.07.99.СС	Схема структурна класів шару сервісів	1	
9	А3	КПІ.ІП-6316.045440.08.99.КЕ	Креслення вигляду екранних форм	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-6316.045440.00.90		
Розробн.		Кошовець Є.П.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Нечай Д.О					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-63	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 51 таблицю, 17 рисунків, 12 джерел.

Об'єкт дослідження: веб-застосування для організації рейтингових турнірів.

Мета дипломного проєкту: автоматизація процесу організації турнірів(тобто створення, проведення жеребкування, підрахунок зміни рейтингу та положення команд у турнірі) для дисциплін у яких в одній грі приймає участь 2 команди або учасника.

У першому розділі було проаналізовано предметну область, доступні технічні рішення та аналоги. Описано варіанти використання, функціональні та нефункціональні вимоги.

У другому розділі було проаналізовано бізнес-процеси, спроектовано архітектуру програмного забезпечення, спроектовано структуру баз даних та обрано СКБД. Обрано мову програмування та інструменти для розробки, проаналізовано безпеку даних та впроваджено механізми для захисту інформації.

У третьому розділі проведено тестування застосунку за розробленим планом тестування. Описано процес тестування.

У четвертому розділі описано розгортання серверної та клієнтської частин додатку, а також наведено інструкції користувача.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗАЦІЯ, ТУРНІР, SPRING, MVC, JAVA, РЕЙТИНГ

ABSTRACT

Explanatory note of the diploma project consists of 4 sections, 51 tables, 17 figures, 12 sources.

The object of study: web-application for organizing ranked tournaments.

The aim of the diploma project: automation of the process of organizing ranked tournaments (creation, drawing pairs, calculation of rating and position of teams/participants changes) for disciplines with two competitive sides.

In the first section, the domain, available technical solutions and alternatives were analyzed. Use-cases, functional and non-functional requirements were described.

In the second section, business processes were analyzed, software architecture was designed, database structure was designed and DBMS was selected. The programming language and development tools have been selected, data security has been analyzed and data security mechanisms have been implemented.

In the third section, application was tested, using testplan. Process of testing was described.

The fourth section describes the deployment of server- and client-sides of the application, as well as user instructions.

KEYWORDS: AUTOMATION, TOURNAMENT, SPRING, MVC, JAVA, RATING

Пояснювальна записка
до дипломного проєкту

на тему: «Веб-застосування для організації рейтингових турнірів з жеребкуванням та двома конкуруючими сторонами»

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення.

Фреймворк – готовий до використання комплекс програмних рішень[1].

Mac OS – операційною системою, створеною компанією Apple Computer[2].

ВСТУП

Кожного дня ми зіштовхуємося із сотнями, тисячами прикладів реалізації різноманітних видів програмного забезпечення. Від тостера до автоматизації великого виробництва – усюди людство використовує ті чи інші аналітичні підходи.

В наш час однією із найбільших сфер є спорт. Вона грає велику роль у житті кожної людини. Спорт – зміцнює здоров'я, робить людину сильнішою, розвиває витривалість, силу волі, закаляє характер та організм. Важко уявити цю область нашого життя без згадування інформаційних технологій.

Спорт неможливо уявити без проведення турнірів. Змагання є ціллю, засобом, методом та моделлю підготовки у будь-якому виді спорту чи іншій діяльності. Це те, що мотивує людей розвиватися та знаходити нові знайомства. Проведення та прийняття участі у турнірах відіграє важливу роль у розвитку кожної людини.

В більшості дитячих гуртків, аматорських та пів-професійних спортивних організаціях будь-якого розміру організування ліги, кубку та жеребкування їхніх турів - не автоматизоване, що призводить до великої кіпи паперу, що зберігає результати турніру, ручного підрахунку положення та зміни рейтингу команди після кожного туру, вибір наступного супротивника. Все це впливає на втрату двох дуже важливих ресурсів: часу та грошей.

Сучасні популярні рішення пропонують автоматизований процес проведення турнірів, але всі вони чи орієнтовані на великі професійні ліги, тобто системи заточені під конкретну асоціацію і турнір, чи реалізують підтримку проведення турнірів без підрахунку змін рейтингу команд та при наявності тільки ручного жеребкування.

Мета створення даної роботи — автоматизація процесу організації турнірів(тобто створення, проведення жеребкування, підрахунок зміни рейтингу та положення команд у турнірі) для дисциплін у яких в одній грі приймає участь 2 команди або учасника.

Завданням даної роботи є дослідження та вибір оптимальних існуючих систем підрахунку рейтингу, організації турнірів та жеребкування, розробка системи для проведення турніру, яку буде зручно налаштовувати для різних видів спорту із двома учасниками. Результатом роботи є система, у якій можна створювати свої асоціації та проводити в них турніри для дисциплін із двома супротивниками.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Турнір – це змагання по якомусь виду спорту/гри, в якому приймає участь відносно велика кількість команд/учасників.

Роль турнірів у розвитку спорту важко переоцінити, точніше кажучи, важко взагалі уявити будь-яку дисципліну без турнірів. Для багатьох людей – це не лише можливість гарно провести час у ролі учасника чи спостерігача, але й привід перевірити рівень своїх навичок та отримати практичний досвід, без якого неможливо уявити розвиток гравця. Беручи всі ці аспекти до уваги, можна ще раз переконатись, що спорт та ігри важко уявити без турнірів, які мотивують учасників та є рушієм їхнього професійного росту.[3]

Турніри можна поділити за місцем та часом проведення на такі види:

– змагання або група змагань, які проходять в одному місці і відбуваються за відносно короткий проміжок часу(наприклад, турнір з гольфу);

– змагання, які розділені на матчі, в кожному з яких приймає участь обмежений набір команд/учасників. Результат такого турніру розраховується на основі результатів індивідуальних матчів.[4]

Другий вид турнірів дуже широко розповсюджений, його особливість в можливості обмеження кількості гравців в одному матчі. Найпопулярнішим підвидом є змагання із матчами, в яких приймає участь 2 конкуруючі сторони. У цього підвиду є безліч прикладів: футбол, шахи , баскетбол, теніс, хокей, багато комп’ютерних та настільних ігор тощо.

Структура проведення турніру з двома гравцями та визначення результату цього турніру залежать від наступних факторів:

– наявність чи відсутність нічиїх у матчі;

– кількість очок, отриманих за перемогу/нічию/поразку;

– формат проведення турніру;

– наявність чи відсутність переваги домашнього поля;

– кількість матчів в одному раунді.

Узагальнивши всі перераховані фактори, можна створити ПЗ, яке буде придатними для проведення турніру із двома конкуруючими сторонами для різних дисциплін.

1.2 Змістовний опис і аналіз предметної області

1.2.1 Рейтингова система

Рейтингова система представляє собою систему, яка аналізує результати змагань, для забезпечення рейтингу команд чи гравців.

Загальні системи включають в себе опитування експертів, не-експертів, букмекерських ринків і комп'ютерних систем. Оцінки, або номінальні потужності, чисельні уявлення конкурентоспроможності часто можна порівняти з тим, що результат гри між будь-якими двома групами може бути передбачений. Рейтингові системи є альтернативою традиційного спортивного розподілу місць команд, яке базується лише на кількості перемог та поразок.[5]

Немає формалізованого розподілу існуючих рейтингових систем на види. Але її можна поділити за факторами, які вони враховують:

– призовий фактор — при підрахунку рейтингу одним із коефіцієнтів є призовий фонд турніру, чим вища нагорода – тим вагоміший коефіцієнт;

– фактор переваги сторони – коли при підрахунку рейтингу важливо, першим чи другим номером грає команда, наприклад білий колір у шахах, домашнє поле у футболі;

– фактор часу – система враховує зміни рейтингу за певний проміжок часу, наприклад за останні 5 років, або може враховувати рейтинг за увесь час одразу;

– фактор набору очок – існують системи, які враховують зміну очок у кожному матчі та системи, які надають пункти рейтингу відповідно до зайнятого у турнірі місця;

Існує 2 основні рейтингові системи : Ело та Гліко у різних варіаціях.

1.2.2 Рейтингова система Гліко

Система Гліко є удосконаленням системи Ело. Головна відмінність полягає у тому, що кожний учасник у таблиці рейтингу, окрім основного рейтингу має оцінку відхилення. Тобто насправді значення рейтингу буде знаходитись у діапазоні, а не являтися одиничним числом.

Така система є актуальною для ігр, де гравці не бачать своїх очок рейтингу, а мають тільки ранг, наприклад популярна комп'ютерна гра Counter-Strike: Global Offensive. Вона враховує кількість матчів, які провів гравець, тобто чим більше матчів - тим менше коефіцієнт відхилення, бо рейтинг стає більш точний. Також на коефіцієнт відхилення може впливати часова затримка між матчами.

1.2.3 Рейтингова система Ело

Система Ело є найбільш популярною рейтинговою системою. З минулого століття її використовують, як базову для більшості видів спорту із двома супротивниками в одному матчі. Футбол, баскетбол, шахи, теніс тощо – усі ці дисципліни використовують систему Ело у тому чи іншому вигляді. Звісно з часом кожна породжена рейтингова система змінювалась відповідно до потреби видів спорту, що вилилось у створення факторів описаних вище, що постають у виді коефіцієнтів у підрахунку рейтингу кожного учасника. Оскільки система Ело досить гнучка, стабільна і підходить для дисциплін з двома гравцями, то будемо використовувати саме її.

Рейтинг Ело не може опускатися нижче 0, при цьому немає верхньої межі, але важко відірватися дуже сильно від другого місця. Адже сильний гравець отримає більше очок від перемоги над слабим, ніж навпаки. Базовий рейтинг, який надається гравцям становить 1400 рейтингових очок.

Оскільки дане програмне забезпечення використовується для видів спорту/ ігор із двома супротивниками у загальному вигляді, а фактори з'явилися із потребою в адаптації рейтингової системи під конкретні дисципліни, то система

Ело буде використовуватись у базовому варіанті.

Зміна рейтингу підраховується в два етапи: знаходиться математичне сподівання (1.1), відбувається підрахунок нового рейтингу учасника(1.2).

Формула підрахунку математичного сподівання:

$$E_A = \frac{1}{1 + 10^{\frac{R_B - R_A}{400}}} \qquad (1.1)$$

, де E_A – математичне сподівання, кількості очок, які набере гравець А в матчі проти гравця В;

R_A – рейтинг гравця А;

R_B – рейтинг гравця В.

Новий рейтинг рахується за наступною формулою:

$$R'_A = R_A + K * (S_A - E_A) \qquad (1.2)$$

, де R'_A – новий рейтинг гравця А;

R_A – старий рейтинг гравця А;

К – коефіцієнт, який набуває значення максимальної можливої зміни рейтингу і набуває значення 16;

S_A – результат матчу(перемога -1, нічия – 0,5, поразка – 0);

E_A – математичне сподівання гравця А.

1.2.4 Формати турнірів та жеребкування матчів

Існує велика кількість різноманітних форматів турніру, ось основні із них:

– кругова система(Ліга): кожен гравець обов’язково грає щонайменше один раз із кожним опонентом. Зазвичай проводиться в один чи два круги;

– швейцарська система: учасники у кожному наступному турі після першого зустрічаються з опонентами, які показали приблизно той самий результат за підсумком попередніх матчів. При цьому важливим є те, що супротивники не можуть повторюватись;

– італійська система – те саме, що і Швейцарська система, але супротивники можуть повторюватись;

– схвенингенська система : команди розбиваються на дві групи і кожен учасник однієї групи грає з усіма учасниками іншої групи;

– олімпійська(нокаут) система: формат у якому команда вибуває після першої поразки. Можливий матч за 3 місце між командами, які вибули у півфіналі;

– дабл-елімінейшн(англ. Double elimination) – аналогічна система до олімпійської, відрізняється можливості двох поразок;

– формат групи + нокаут: система проведення турніру, яка складається з двох етапів: в першому з яких команди розбиваються на групи і в кожній команди грають по круговій системі, а другий етап – нокаут турнір.

Найбільш розповсюдженні формати, які і були обрани для використання у програмному забезпеченні: швейцарська система, кругова система, нокаут система.

При проведені турнірів важливу роль займає жеребкування матчів між раундами. Воно має відповідати особливостям турніру.

Нокаут система використовується, коли треба за мінімальну кількість матчів знайти переможця турніру. Перш за все кількість учасників має бути степенем двійки, при іншому числі треба провести додатковий раунд серед найслабших за рейтингом команд. Коли в нас є правильна кількість команд і треба обрати спосіб підбору команд у матчі. Можна сформувати раунди так, щоб найсильніші за рейтингом учасники попадали у пари із слабкими. Таким чином з великою вірогідністю кожна команда досягне раунд приблизно вартий свого рейтингу, але при цьому пропадає інтерес до перших етапів, де результат майже очевидний і слабким командам не надається шанс проявити себе. При цьому, якщо створювати пари, у яких сильні учасники змагаються з сильними, а слабкі із слабкими , то отримаємо у фіналі матч із командою набагато слабшою за іншого фіналіста. Отже, задля підтримки інтриги на всіх етапах у кожному раунді команди будуть підібрані випадковим чином.

Кругова система – це зазвичай довгий турнір, який часто проводиться в 2

круги, щоб уникнути переваги домашнього поля одного із учасників. Оскільки ми знаємо, що кожний учасник грає із кожним, то єдине, що потрібно враховувати, щоб ніхто із учасників не залишився без пари у наступному турі, випадком є ситуація із непарною кількістю гравців, де обов'язково є одна команда, яка відпочиває у турі. Жеребкування проводиться у відповідності із назвою, ділимо команди на 2 рядки і пишемо їх відповідний номер, додаємо 0 при непарній кількості команд, далі учасники в одному стовпці і є парою турі, в наступних турах залишаємо першу команду на місці і за годинниковою стрілкою рухаємо всі інші номери. Таким чином зберігаються умови, що кожна команда зіграє із кожною.

Швейцарська система була створена за умови, що турнір за круговою системою дуже довгий, а за нокаут системою об'єктивно визначається лише перше місце в турнірі. Швейцарська система була створена, щоб розподіл місць всього турніру був об'єктивний за оптимальну кількість турів. Кількість турів визначається за формулою. Жеребкування проводиться окремо для першого раунду та для всіх інших. Перший тур – команди сортуються за рейтингом і діляться умовно навпіл, таким чином утворюються матчі, де перший гравець з верхньої половини таблиці зустрічається із першим гравцем з нижньої частини, другий із другим і так далі. Далі таким самим чином, тільки гравці розбиваються на групи за кількістю набраних очок. В кінці місця при однаковій кількості набраних очок розподіляються за коефіцієнтом Бухгольца.

1.3 Аналіз успішних ІТ-проєктів

1.3.1 Аналіз відомих технічних рішень

Існує велика кількість програмних застосувань, які надають користувачам можливість створюють та проводити турніри. Багато організацій не поєднаних напряму зі спортом проводять різноманітні турніри між співробітниками або між друзями тощо. Такій аудиторії немає сенсу витрачати величезні гроші на створення спеціалізованої строго під них системи, при цьому вони не проти

платити відносно невеликі гроші за надання сервісу, де надається достатньо можливостей для проведення турнірів та ліг без зайвих клопіт.

Є 2 популярні підходи для реалізації цієї проблеми:

– сайти-конструктори: сервіси, в яких ти створюєш окремий сайт саме під свій турнір чи асоціацію, кастомізуючи блоки на сторінці;

– сайт – база: сервіс, де багато користувачів одночасно створюють і проводять велику кількість турнірів з різноманітних видів спорту.

В обох випадках майже завжди сервіси мають платні та безкоштовні плани користувача, але безкоштовний план зазвичай надто урізаний і практично унеможливлює його використання, обмежуючи кількість команд та турів, проте надає доступ до дуже детальної статистики та корегувань деталей(наприклад форма команди).

Більшість рішень розраховують положення команд в рамках одного конкретного турніру. Рейтинг на великій дистанції не враховується. Майже на всіх сайтах один перелік форматів, який майже завжди включає кругову та нокаут системи. Жеребкування проводиться вручну чи випадковим чином.

1.3.2 Аналіз відомих програмних продуктів

Популярний представник сайтів-конструкторів: **LeagueRepublic.**

У цього сайту зручний та приємний інтерфейс. При оплаті можна зберігати дуже детальну статистику. Серед значних переваг: при створенні можна в один клік генерувати розклад на всю лігу , який може враховувати різну кількість факторів(наприклад, уникнення конфлікту при поділі місця проведення матча різними командами).Є онлайн підтримка, яка допоможе при виникненні якихось питань стосовно керування турніру. Надає можливості контролю внесків чи інших переводів грошей учасниками команд. Наявні колонки новин.

Головні недоліки : безкоштовно працює тільки 14 днів і має ряд обмежень, відсутність загального рейтингування команд, є автоматичне жеребкування, але фактично воно враховує тільки людські фактори, а не рейтинг.

Популярний представник сайту-бази: **Konkuri.**

Основні особливості:

– можливо створити турнір у 3х форматах(гра на вибування, група та їх поєднання);

– можливе представлення учасників у вигляді однієї особи, дабла(команда із 2х осіб) чи просто команди(яка складається з окремих учасників;

– жеребкування проводиться випадковим чином або вручну;

– наявна нотифікація результатів через email;

– приємний інтерфейс;

– при створенні турніру можна вибрати 1 із багатьох заготовлених видів спорту/гри, що допомагає надавати актуальнішу статистику;

– відсутність рейтингової системи поза межами одного турніру;

– практична неможливість використання сайту безкоштовно(максимум команд на турнір – 4).

1.4 Аналіз вимог до програмного забезпечення

Для визначення вимог до програмного забезпечення необхідно визначити ролі користувачів та їхні можливості в системі. Для досягнення мети розробки система повинна містити наступні типи користувачів:

– організатор;

– користувач.

Користувач – це неавторизований користувач, він має можливість переглядати турніри, асоціації та положення команд у рейтингу асоціації.

Організатор – це авторизований користувач, він має такі ж можливості, що і користувач, а також він може створювати асоціації, турніри та команди.

Загалом система повинна мати такий функціонал:

– авторизація користувача;

– реєстрація користувача;

– перегляд профілю організатора;

– перегляд профіля асоціації;
– перегляд положення команд в рейтинговій таблиці асоціації;
– створення асоціацій;
– створення турніру в асоціації;
– створення команди/учасника в асоціації;
– проведення турніру;
– додавання команди до турніру;
– підрахунок зміни рейтингу команд.

1.4.1 Розроблення функціональних вимог

Детальну схему варіантів використання наведено в документі КПІ.ІП-6316.045440.06.99.СС Схема структурна варіанти використання.

В системі передбачено наступні варіанти використання:

Таблиця 1.1 – Варіант використання UC001

Назва	Реєстрація в системі.
Опис	Користувач реєструється в системі.
Учасники	Користувач.
Передумови	Відкрита сторінка реєстрації користувача.
Постумови	Користувач зареєстрований в системі.
Основний сценарій	Користувач заповнює усі необхідні поля валідними даними. Користувач натискає кнопку реєстрації.
Розширення сценаріїв	

Таблиця 1.2 – Варіант використання UC002

Назва	Авторизація в системі.
Опис	Користувач авторизується в системі.
Учасники	Користувач.

Продовження таблиці 1.2

Передумови	Користувач зареєстрований в системі. Відкрита сторінка авторизації користувача.
Постумови	Користувач авторизувався в системі як організатор.
Основний сценарій	Користувач заповнює усі необхідні поля валідними даними. Користувач натискає кнопку авторизації.
Розширення сценаріїв	

Таблиця 1.3 – Варіант використання UC003

Назва	Створення асоціації.
Опис	Організатор створює свою асоціацію.
Учасники	Організатор.
Передумови	Користувач авторизувався як організатор.
Постумови	Організатор успішно створив асоціацію.
Основний сценарій	Організатор натискає кнопку створення асоціації. Система демонструє вікно створення асоціації. Організатор вводить назву асоціації, назву дисципліни та інформацію про асоціацію. Організатор натискає кнопку створити асоціацію. Система створює асоціацію.
Розширення сценаріїв	

Таблиця 1.4 – Варіант використання UC004

Назва	Створення турніру.
Опис	Організатор створює турнір.

Продовження таблиці 1.4

Учасники	Організатор.
Передумови	Користувач авторизований як організатор і має асоціацію, в асоціації є створенні команди/учасники.
Постумови	Організатор успішно створив турнір.
Основний сценарій	Організатор натискає на кнопку створення турніру. Система показує вікно створення турніру. Організатор у формі вказує назву, опис та формат турніру. Натискає кнопку додати команди/учасників. Система відображає вікно із списком всіх команд/учасників, які були створенні в асоціації. Організатор вибирає команду/учасника і натискає кнопку додати. Коли всі команди/учасники додані організатор натискає кнопку створити турнір. Система створює турнір.
Розширення сценаріїв	

Таблиця 1.5 – Варіант використання UC005

Назва	Створення команди/учасника.
Опис	Організатор додає команду/учасника в асоціацію.
Учасники	Організатор.
Передумови	Користувач авторизувався як організатор та має створену асоціацію.
Постумови	Успішне додавання нової команди/учасника в асоціацію.

Продовження таблиці 1.5

Основний сценарій	Організатор заходить на сторінку асоціації і натискає кнопку додати учасника. Система відображає сторінку створення команди/учасника. Організатор вказує ім’я/назву учасника/команди, інформацію про учасника та його поточний рейтинг. Організатор натискає кнопку створення команди/учасника. Система створює команду/учасника.
Розширення сценаріїв	

Таблиця 1.6 – Варіант використання UC006

Назва	Перегляд свого профілю.
Опис	Організатор переглядає свій профіль.
Учасники	Організатор.
Передумови	Користувач авторизований як організатор.
Постумови	Організатор бачить свій профіль.
Основний сценарій	Організатор натискає кнопку профіля. Система відображає сторінку профіля: ім’я користувача, його інформацію, список створених асоціацій, кнопку створення асоціації.
Розширення сценаріїв	

Таблиця 1.7 – Варіант використання UC007

Назва	Перегляд сторінки своєї асоціації.
Опис	Організатор переглядає сторінку асоціації.

Продовження таблиці 1.7

Учасники	Організатор.
Передумови	Користувач авторизувався як організатор і має створену асоціацію.
Постумови	Організатор бачить сторінку асоціації.
Основний сценарій	Організатор заходить на свій профіль і натискає на кнопку переходу до профіля відповідної асоціації. Система показує сторінку асоціації: назву асоціації, її дисципліни, інформацію про асоціацію, список створених в асоціації турнірів, кнопку створення турніру, кнопку створення учасника, кнопку перегляду таблицю рейтингу учасників.
Розширення сценаріїв	

Таблиця 1.8 – Варіант використання UC008

Назва	Перегляд рейтингу команд в асоціації.
Опис	Користувач переглядає рейтинг команд в асоціації .
Учасники	Користувач.
Передумови	Асоціація існує.
Постумови	Користувач бачить таблицю рейтингу асоціації.

Продовження таблиці 1.8

Основний сценарій	Користувач переходить на сторінку пошуку асоціацій. Система відображає список всіх користувачів. У рядку пошуку вписує назву асоціації. Натискає кнопку пошуку. Система показує сторінку із списком асоціацій із відповідними назвами. Користувач вибирає відповідну асоціацію та переходить до сторінки асоціації натиснувши відповідну кнопку. Система показує сторінку асоціації. Користувач натискає кнопку таблиця рейтингу. Система показаю сторінку таблиці рейтингу.
Розширення сценаріїв	

Таблиця 1.9 – Варіант використання UC009

Назва	Перегляд профілю турніру.
Опис	Користувач переглядає профіль турніру.
Учасники	Користувач.
Передумови	Турнір існує.
Постумови	Користувач бачить профіль турніру.
Основний сценарій	Користувач заходить на сорінку пошуку турніру. Система відображає всі існуючі турніри. Користувач у рядку пошуку вписує назву турніру Натискає кнопку пошуку. Система показує сторінку із списком турнірів із відповідними назвами. Користувач вибирає відповідний турнір та переходить до сторінки турніру натиснувши відповідну кнопку.

Продовження таблиці 1.9

Розширення сценаріїв	

Таблиця 1.10 – Варіант використання UC010

Назва	Перегляд сторінки свого турніру.
Опис	Організатор переглядає сторінку свого турніру.
Учасники	Організатор.
Передумови	Користувач авторизувався ,як організатор, має створену асоціацію і створений в ній турнір.
Постумови	Організатор бачить сторінку турніру.
Основний сценарій	Організатор заходить на свій профіль і натискає на кнопку переходу до сторінки відповідної асоціації. Система показує сторінку асоціації. Користувач вибирає відповідний турнір і натискає на кнопку перейти до сторінки турніру. Система відображає сторінку турніру: назву, опис, формат турніру, таблицю положення команд/учасників у турнірі, та кнопки в залежності від стану турніру(скоро почнеться, відбувається, закінчений).
Розширення сценаріїв	

Таблиця 1.11 – Варіант використання UC011

Назва	Перегляд профілю асоціації.
Опис	Користувач переглядає профіль асоціації.
Учасники	Користувач.
Передумови	Існує асоціація.
Постумови	Користувач бачить профіль асоціації.

Продовження таблиці 1.11

Основний сценарій	Користувач переходить на сторінку пошуку асоціацій. Користувач у рядку пошуку вписує назву асоціації та вибирає пошук за назвою. Натискає кнопку пошуку. Система показує сторінку із списком асоціацій із відповідними назвами. Користувач вибирає відповідну асоціацію та переходить до сторінки асоціації натиснувши відповідну кнопку. Система показує сторінку асоціації.
Розширення сценаріїв	

Таблиця 1.12 – Варіант використання UC012

Назва	Перегляд профілю організатора.
Опис	Користувач переглядає профіль організатора.
Учасники	Користувач.
Передумови	Організатор існує.
Постумови	Користувач бачить профіль організатора, якого шукав.
Основний сценарій	Користувач у рядку пошуку вписує логін організатора і вибирає пошук за логіном. Натискає кнопку пошуку. Система показує сторінку із профілем відповідного організатора.
Розширення сценаріїв	

Таблиця 1.13 – Варіант використання UC013

Назва	Розпочати турнір.

Продовження таблиці 1.13

Опис	Авторизований користувач розпочинає створений турнір.
Учасники	Організатор.
Передумови	Користувач авторизувався ,як організатор, має створену асоціацію і створений в ній турнір. Турнір ще не почався.
Постумови	Турнір знаходиться в стані «Відбувається».
Основний сценарій	Організатор переходить на профіль турніру. Система показає сторінку турніру. Організатор натискає кнопку розпочати турнір. Система отримує повідомлення про початок турніру.
Розширення сценаріїв	

Таблиця 1.14 – Варіант використання UC014

Назва	Закінчити раунд.
Опис	Організатор закінчує раунд в турнірі.
Учасники	Організатор.
Передумови	Користувач авторизований ,як організатор, має створену асоціацію і створений в ній турнір. Турнір знаходиться в стані «Відбувається». Проходить не останній тур турніру.
Постумови	Збережено результат раунду.
Основний сценарій	Організатор переходить на сторінку турніру. Система відображає сторінку турніру. Організатор вписує результати раунду і натискає кнопку закінчити раунд. Система зберігає результати раунду.

Продовження таблиці 1.14

Розширення сценаріїв	

Таблиця 1.15 – Варіант використання UC015

Назва	Закінчення турніру.
Опис	Організатор закінчує турнір.
Учасники	Організатор.
Передумови	Користувач авторизований, як організатор. В турнірі завершились всі раунди.
Постумови	Турнір завершений.
Основний сценарій	Організатор заходить на сторінку відповідного турніру. Система відображає сторінку турніру. Користувач натискає кнопку «Завершити турнір». Система змінює статус турніру з «Відбувається» на «Завершений».
Розширення сценаріїв	

Функціональні вимоги описані наступними таблицями:

Таблиця 1.16 – Опис функціональної вимоги REQ001

Номер	REQ001
Назва	Реєстрація в системі.
Опис	Система має надавати неавторизованому користувачу можливість реєстрації.

Таблиця 1.17 – Опис функціональної вимоги REQ002

Номер	REQ002
Назва	Авторизація в системі.
Опис	Система має зберігати дані користувача зареєстрованими, і надавати йому можливість авторизації через логін і пароль.

Таблиця 1.18 – Опис функціональної вимоги REQ003

Номер	REQ003
Назва	Створення асоціації.
Опис	Створення асоціації можливе тільки користувачу, авторизованому як організатор.

Таблиця 1.19 – Опис функціональної вимоги REQ004

Номер	REQ004
Назва	Створення турніру.
Опис	Система надає можливість створювати турнір лише в межах асоціації, створеної користувачем, авторизованим як організатор. Команд/учасників має бути від 2 до 64.

Таблиця 1.20 – Опис функціональної вимоги REQ005

Номер	REQ005
Назва	Додавання команди/учасника в асоціацію.
Опис	Система надає можливість створювати команду/учасника лише в межах асоціації, створеної авторизованим користувачем.

Таблиця 1.21 – Опис функціональної вимоги REQ006

Номер	REQ006
Назва	Перегляд власного профілю.
Опис	Система відібражає власний профіль у вигляді: інформації про організатора, списку його асоціацій та кнопки створення нової асоціації. Відображення можливе, коли користувач, авторизований як організатор.

Таблиця 1.22 – Опис функціональної вимоги REQ007

Номер	REQ007
Назва	Перегляд власної асоціації.
Опис	Система відображає сторінку асоціації у вигляді: інформації про асоціацію, списку створених в ній турнірів, кнопок створення нового учасника, турніру та переходу до таблиці рейтингу .Таке відображення можливе, коли користувач. Авторизований як організатор і має створені асоціації.

Таблиця 1.23 – Опис функціональної вимоги REQ008

Номер	REQ008
Назва	Перегляд власного турніру.
Опис	Система відображає власний турнір у вигляді: інформації про турнір, таблиці статистики та залежно від статусу турніру різні кнопки керування та результати раундів(При статусі «Скоро почнеться»: кнопка «Розпочати турнір» та ніяких результатів, при статусі «Відбувається»: кнопка закінчити раунд, можливість додати результати поточного раунду, відображення результату попереднього раунду).

Таблиця 1.24 – Опис функціональної вимоги REQ009

Номер	REQ009
Назва	Пошук організатора.
Опис	Система має можливість пошуку організаторів. Пошук відбувається за логіном організатора та видає результат у вигляді всіх організаторів, логіни яких співпадають із заданим значенням.

Таблиця 1.25 – Опис функціональної вимоги REQ010

Номер	REQ010
Назва	Пошук асоціації.
Опис	Система має можливість пошуку асоціацій. Пошук відбувається за назвою асоціації та видає результат у вигляді всіх асоціацій, назви яких співпадають із заданим значенням.

Таблиця 1.26 – Опис функціональної вимоги REQ011

Номер	REQ011
Назва	Пошук турніру.
Опис	Система має можливість пошуку турнірів. Пошук відбувається за назвою турніру та видає результат у вигляді всіх турнірів, назви яких співпадають із заданим.

Таблиця 1.27 – Опис функціональної вимоги REQ012

Номер	REQ012
Назва	Перегляд профілів.
Опис	Система надає можливість переглядати профілі організаторів. Вона подає їх у наступному вигляді: інформація про користувача(його ім’я, коротка інформація про нього), список асоціацій організатора .

Таблиця 1.28 – Опис функціональної вимоги REQ013

Номер	REQ013
Назва	Перегляд асоціацій .
Опис	Система надає можливість переглядати асоціації всіх організаторів. Вона подає їх у наступному вигляді: інформація про асоціацію, списку створених в ній турнірів, та кнопки переходу до таблиці рейтингу.

Таблиця 1.29 – Опис функціональної вимоги REQ014

Номер	REQ014
Назва	Перегляд турнірів.
Опис	Система надає можливість переглядати турніри всіх організаторів. Вона подає їх у наступному вигляді: інформація про турнір, таблиці статистики та в залежності від стану турніру показувати чи не показувати результати раундів, які вже відбулися.

Таблиця 1.30 – Опис функціональної вимоги REQ015

Номер	REQ015
Назва	Підрахунок зміни рейтингу.
Опис	Система рахує зміну рейтингу команд/учасників. Підрахунок має відбуватися після закінчення кожного раунду.

Таблиця 1.31 – Опис функціональної вимоги REQ016

Номер	REQ016
Назва	Початок турніру.

Продовження таблиці 1.31

Опис	Користувач має можливість розпочати турнір натиснувши кнопку «Розпочати турнір» на сторінці турніру. Турнір має знаходитися у стані «Скоро розпочнеться», користувач має бути авторизований як організатор.

Таблиця 1.32 – Опис функціональної вимоги REQ017

Номер	REQ017
Назва	Закінчення раунду.
Опис	Організатор має можливість закінчити раунд в турнірі натиснувши кнопку «Закінчити раунд» на сторінці турніру. Турнір має знаходитися у стані «Відбувається», користувач має бути авторизований як організатор, повинні бути внесені всі результати раунду.

Таблиця 1.33 – Опис функціональної вимоги REQ018

Номер	REQ018
Назва	Жеребкування раунду.
Опис	Система має автоматично проводити жеребкування турнірів та розподіл учасників на матчі після початку турніру чи закінчення одного із його раундів, крім останнього.

Таблиця 1.34 – Опис функціональної вимоги REQ019

Номер	REQ019
Назва	Закінчення турніру.

Продовження таблиці 1.34

Опис	Організатор має можливість закінчити турнір, натиснувши кнопку «Закінчити турнір» на сторінці турніру. Турнір має знаходитися у стані «Відбувається», повинен бути закінченим останній раунд, користувач має бути авторизований як організатор.

Взаємозв'язки між вимогами і варіантами використання відображено на рисунку 1.1.

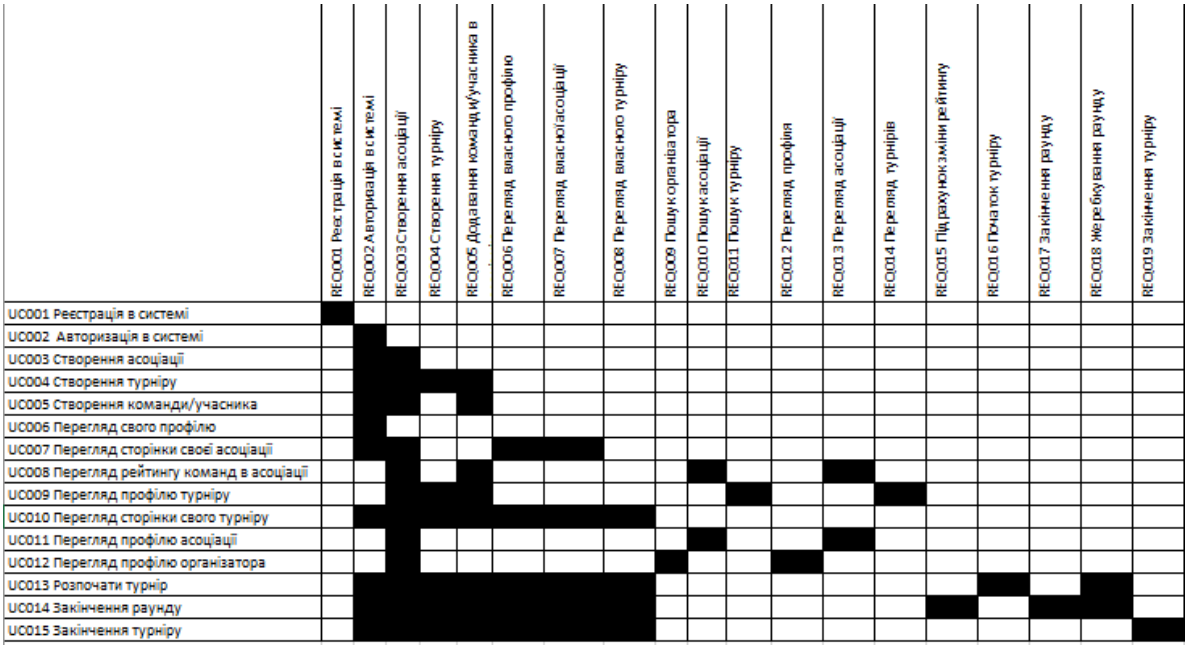


Рисунок 1.1 – Матриця залежності між вимогами застосунку і варіантами використання

1.4.2 Розроблення нефункціональних вимог

Програмне забезпечення повинне відповідати наступним нефункціональним вимогам:

– мова додатку - англійська;

– працювати в наступних браузерах: Google Chrome, Internet Explorer, Mozilla Firefox, Safari, Opera Browser;

– працювати на Windows OS.

1.4.3 Постановка комплексу завдань модулю

Ціллю розробки є автоматизація процесу організації турнірів(тобто створення, проведення жеребкування, підрахунок зміни рейтингу та положення команд у турнірі) для дисциплін у яких в одній грі приймає участь 2 команди або учасника.

При аналізі створених функціональних та нефункціональних вимог та варіантів використання були сформульовані наступні задачі, що має вирішити розроблене програмне забезпечення:

– надати можливість користувачу створювати обліковий запис;

– надати можливість користувачу створювати асоціації, додавати турніри та команди/учасників в асоціаціях;

– надати можливість користувачу переглядати асоціації та турніри;

– надати можливість користувачу пошуку асоціацій та турнірів.

– коректний підрахунок зміни рейтингу команд/учасників після матчів;

– коректне жеребкування при використанні нокаут системи, кругової системи, швейцарської системи;

– надати можливість користувачу проводити турніри.

1.5 Висновки по розділу

У цьому розділі було оглянуто поняття турніру, його значення в сфері спорту та ігор. Було оглянуто 2 основні види турнірів. Було описано підвид турнірів із двома конкуруючими сторонами та ряд факторів, які впливають на структуру цих змагань.

Відбулося ознайомлення із поняттям рейтингової системи, факторами за якими відрізняють різні рейтингові системи.

Було розглянуто основні рейтингові системи: Гліко та Ело. Для розробки ПЗ була вибрана система рейтингу Ело у базовому вигляді.

Було розглянуто поняття формату турніру. Були описані найпопулярніші

формати проведення турніру. Серед них було відібрано 3 основні формати та описано для кожного із них вибране жеребкування учасників для кожного раунду, яке відповідає особливостям кожного із форматів.

Були розглянуті та проаналізовані 2 основні направлення технологій побудування програм-аналогів. Біли розглянуті програми – аналоги. Будо визначено їх головні недоліки, а саме : відсутність підтримки рейтингової системи та відсутність автоматичного жеребкування чи його наявність, але без використання алгоритмів.

Були детально розписані функціональні та нефункціональні вимоги ПЗ, на їх основі побудовано матрицю трасування.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Необхідне детальне моделювання архітектури і процесів програмного забезпечення для розробки якісного продукту. Будемо використовувати методологію створення діаграм BPMN для проєктування бізнес-процесів. Основні бізнес-процеси, що відбуваються в системі:

— реєстрація користувача;
— авторизація користувача;
— створення асоціації;
— створення турніру;
— створення команди;
— проведення раунду турніру;
— пошук турніру;
— пошук асоціації;
— пошук користувача.

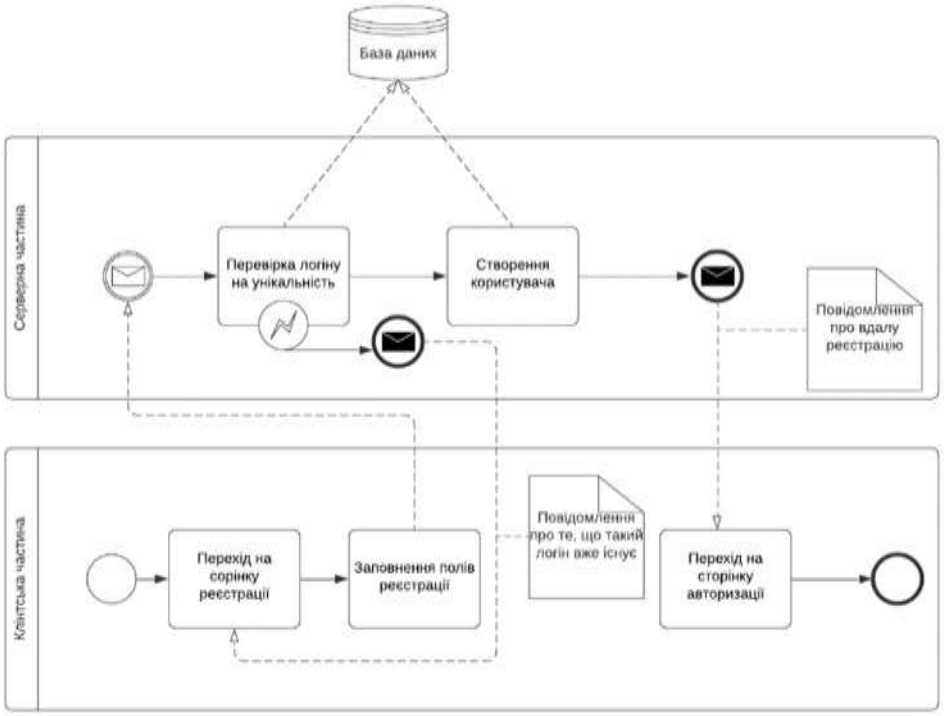


Рисунок 2.1 – Схема бізнес-процесу реєстрація користувача

Послідовний опис реєстрації користувача:

– користувач заходить на сторінку реєстрації;

– користувач заповнює поля форми;

– запит на реєстрацію відправляється на сервер;

– якщо вибраний користувачем логін вже використовується, то система про це повідомляє;

– якщо помилок не виявлено, то система реєструє користувача;

– система відображає сторінку авторизації.

Рисунок 2.2 – Схема бізнес-процесу авторизація користувача

Послідовний опис авторизації користувача:

– користувач заходить на сторінку авторизації;

– користувач вводить логін і пароль;

– дані відправляються на сервер;

– якщо логін чи пароль не співпадають, то система повідомляє про помилку;

– якщо помилок немає, то сесія запам’ятовує користувача і користувач перенаправляється на сторінку свого профіля.

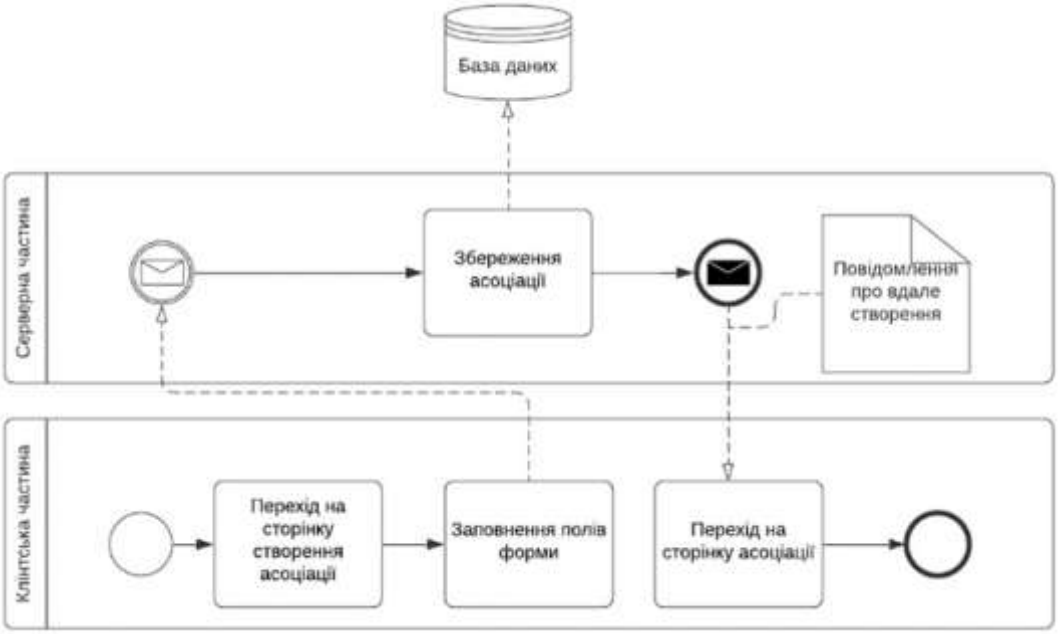


Рисунок 2.3 – Схема бізнес-процесу створення асоціації

Послідовний опис створення асоціації:

– користувач відкриває сторінку створення асоціації;
– користувач заповнює поля форми;
– дані відправляються на сервер;
– асоціація зберігається в базі даних;
– користувач переходить до сторінки створеної асоціації.

Рисунок 2.4 – Схема бізнес-процесу створення учасника

Послідовний опис створення учасника:

– користувач відкриває сторінку асоціації, в якій хоче створити команду;

– система відображає дані асоціації, список створених в ній турнірів;

– користувач переходить до сторінки створення команди;

– користувач заповнює форму створення учасника;

– дані передаються на сервер;

– система підраховує місце команди в рейтингу асоціації;

– команда зберігається в базі даних;

– користувач перенаправляється на сторінку поточної асоціації.

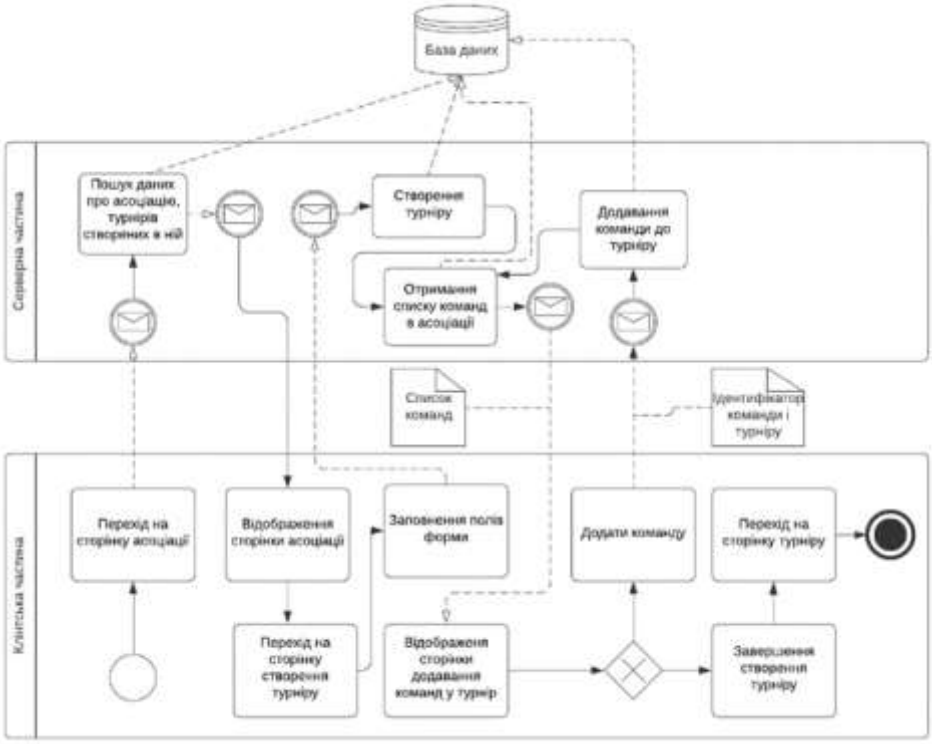


Рисунок 2.5 – Схема бізнес-процесу створення турніру

Послідовний опис створення турніру:

– користувач відкриває сторінку асоціації, в якій хоче створити команду;

– система відображає дані асоціації, список створених в ній турнірів;

– користувач переходить до сторінки створення турніру;

– користувач заповнює форму;

– дані відправляються на сервер;

– сервер зберігає сутність турнір;

– користувач перенаправляється на сторінку додавання команд;

– система відображає список команд, які існують в асоціації;

– користувач може додати команди команду до турніру натиснувши кнопку «Додати» навпроти відповідної команди;

– якщо користувач додав команду, то система додає дану команду до поточного турніру;

– якщо користувач додав потрібні йому команди, він натискає кнопку «Створити турнір»;

– користувач перенаправляється на сторінку турніру.

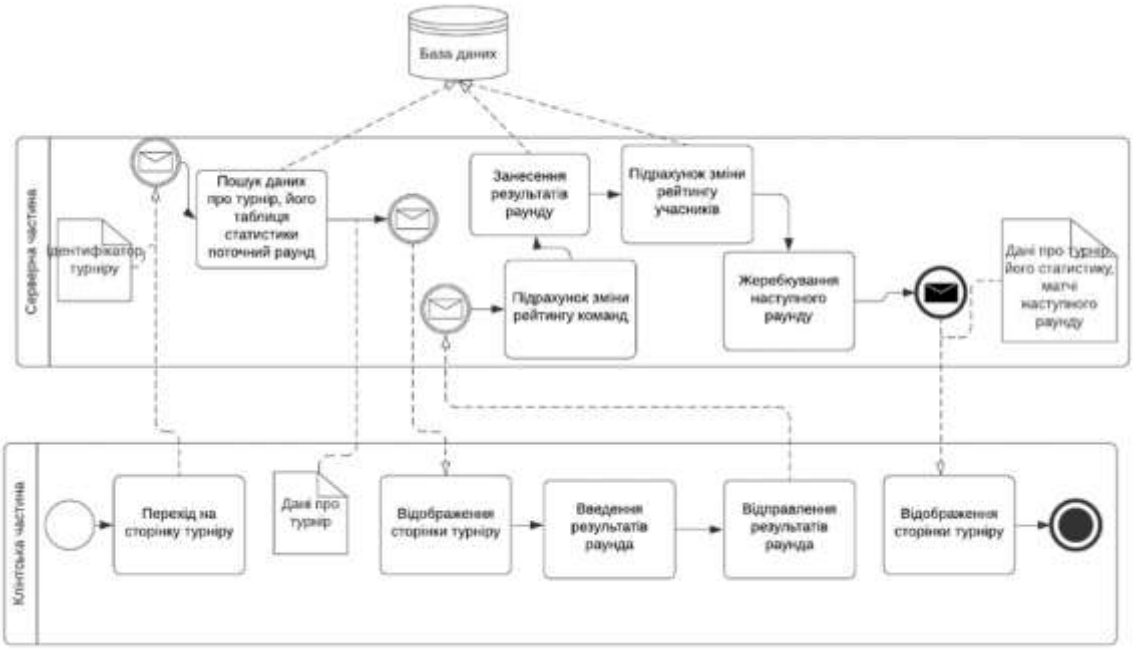


Рисунок 2.6 – Схема бізнес-процесу проведення раунду

Послідовний опис проведення раунду:

– користувач відкриває сторінку турніру;

– вводить результат навпроти кожного матчу і натискає кнопку «Зберегти»;

– дані відправляються на сервер;

– система підраховує рейтинг учасників у зіграних матчах;

– система зберігає результати і оновлює таблицю статистики;

– система проводить жеребкування наступного раунду;

– користувач бачить оновлену сторінку турніру, де може вводити результати уже наступного раунду.

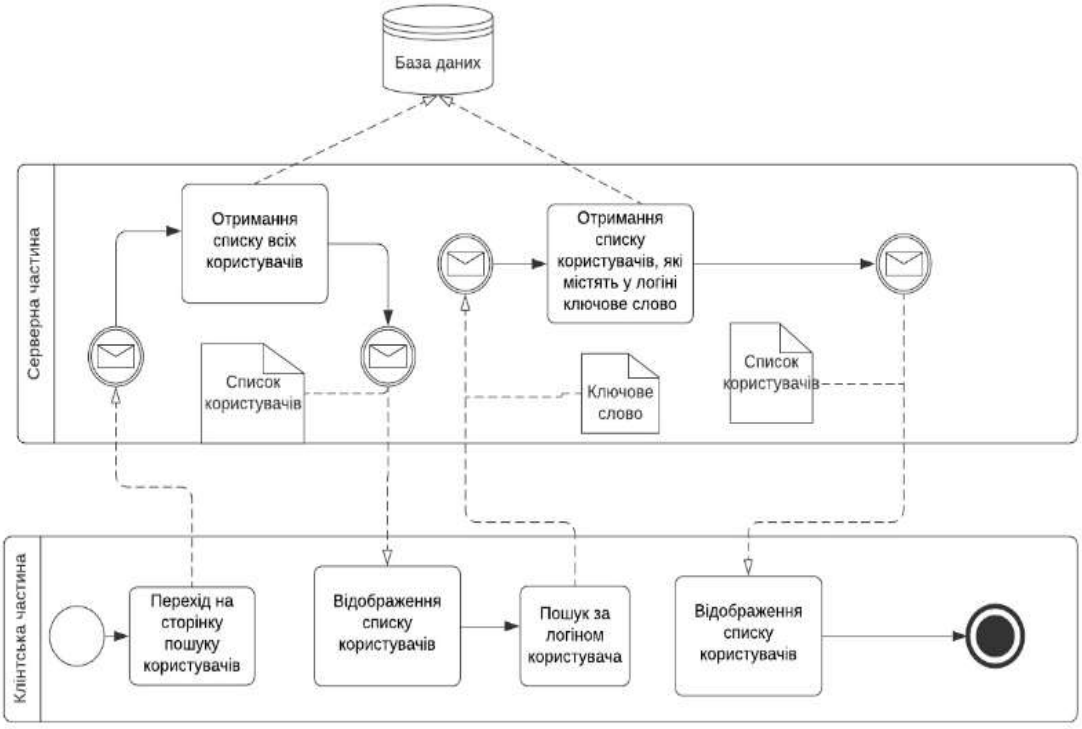


Рисунок 2.7 – Схема бізнес-процесу пошук користувача

Послідовний опис пошуку користувача:

– користувач відкриває сторінку пошуку користувача;

– система відображає список всіх існуючих користувачів;

– користувач вводить у поле пошуку логін користувача, якого він шукає;

– дані відправляються на сервер;

– система шукає в базі даних всіх користувачів, логін яких містить у собі задане користувачем слово;

– система відображає список знайдених користувачів на сторінці пошуку.

Рисунок 2.8 – Схема бізнес-процесу пошук турніру

Послідовний опис пошуку турніру:

– користувач відкриває сторінку пошуку турніру;

– система відображає список всіх існуючих турнірів;

– користувач вводить у поле пошуку назву турніру, якого він шукає;

– дані відправляються на сервер;

– система шукає в базі даних всі турніри, назва яких містить у собі задане користувачем слово;

– система відображає список знайдених турнірів на сторінці пошуку.

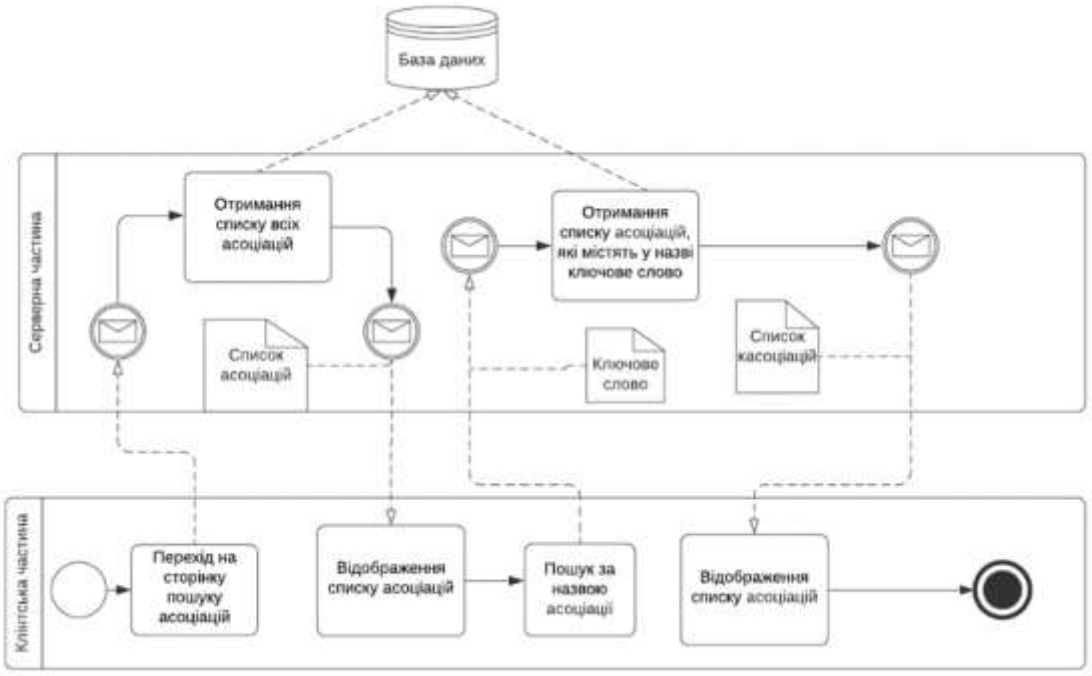


Рисунок 2.9 – Схема бізнес-процесу пошук асоціації

Послідовний опис пошуку асоціації:

– користувач відкриває сторінку пошуку асоціації;

– система відображає список всіх існуючих асоціацій;

– користувач вводить у поле пошуку назву асоціації, яку він шукає;

– дані відправляються на сервер;

– система шукає в базі даних всі асоціації, назва яких містить у собі задане користувачем слово;

– система відображає список знайдених асоціацій на сторінці пошуку.

2.2 Архітектура програмного забезпечення

Перед розробкою ПЗ важливо правильно обрати мову розробки та фреймворки, що будуть використані в процесі розробки.

В якості мови розробки була обрана Java. Вона є платформо-незалежною, що дасть змогу розгорнути веб-застосування як на Windows, так на Linux і MAC OS. Java має високу продуктивність завдяки використання байт-коду, забезпечує надійну роботу при роботі із багатьма потоками.

В якості основного фреймворка був обраний Spring Framework. Це

найпопулярніший фреймворк для розробки Java проєктів. Мільйони розробників використовують Spring Framework для створення високопродуктивного коду, який легко тестується та багаторазово використовується. Це відкрита платформа із Java кодом. Spring Framework спрямований на полегшення розробки веб-додатків.

Spring – це ціла мікросистема, в якій окрім основного великого фрейморку Spring Core є ряд інших фреймворків, які добре взаємодіють один між одним і полегшують роботу при розробці програміста:

– Spring Security – надає можливість гнучкого налаштування автентифікації та контролю доступу;

– Spring Data – надає можливість безпечно та швидко працювати з даними в системі;

– Spring MVC – фреймворк, який забезпечує архітектуру MVC та надає ряд готових компонентів, які можуть використовуватись для гнучкої розробки веб-додатків:

– Spring Boot – фреймворк, який виконує велику кількість налаштувань, зберігаючи час розробника. Він містить у собі Tomcat – контейнер сервлетів, який використовується в якості самостійного веб-сервера.[6]

Візуальне представлення веб – сторінок буде підтримуватись Thymeleaf – сучасним механізмом Java на основі шаблонів для веб.

Для комфортної роботи із базою даних буде використовуватись фрейморк Hibernate – найпопулярніша реалізація специфікації JPA. Цей фреймворк надає деякий абстрактний рівень розробки, що означає повне налаштування задач низького рівня, таких як: встановлення з’єднання із базою даних, написання CRUD-запитів тощо.[7]

В якості бази даних використовується MySQL. Це найпопулярніша реляційна база даних у світі. В неї є наступні важливі переваги :

– безпека даних;

– гнучкість;

– легке масштабування даних;

– висока продуктивність;

– безперебійна робота ;

– виконання ACID.

Програма була створена за архітектурою патерна MVC, де:

– M(model) – модель відповідає за виконання бізнес-логіки та взаємодії з даними;

– V(view) – представлення – відповідає за те що бачить користувач, візуалізацію програми;

– C(Controller) – контролер – є прошарком між моделлю та представленням. Контролює потік даних, що надходить до моделі та оновлює представлення при зміні відповідних даних. Розділяє модель і представлення.

Схема MVC представлена на рисунку 2.10 – Шаблон MVC.

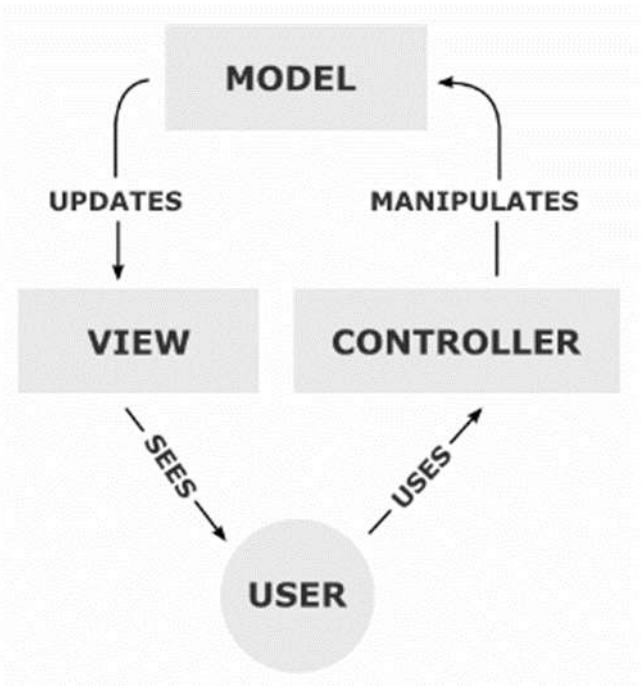


Рисунок 2.10 – Шаблон MVC

В свою чергу Spring MVC містить наступні компоненти:

– Front Controller – головний вбудований контролер, який приймає всі вхідні запити та делегує їх котроллерам користувача, а також повертає відповідь від інших контролерів;

– Controller – контролер користувача, який обробляє запит, якщо той до нього доходить, потім відправляє відповідь Front Controller;

– View template – представлення.

Схема роботи компонентів Spring MVC представлена на рисунку 2.11 – Взаємодія компонентів Spring MVC.

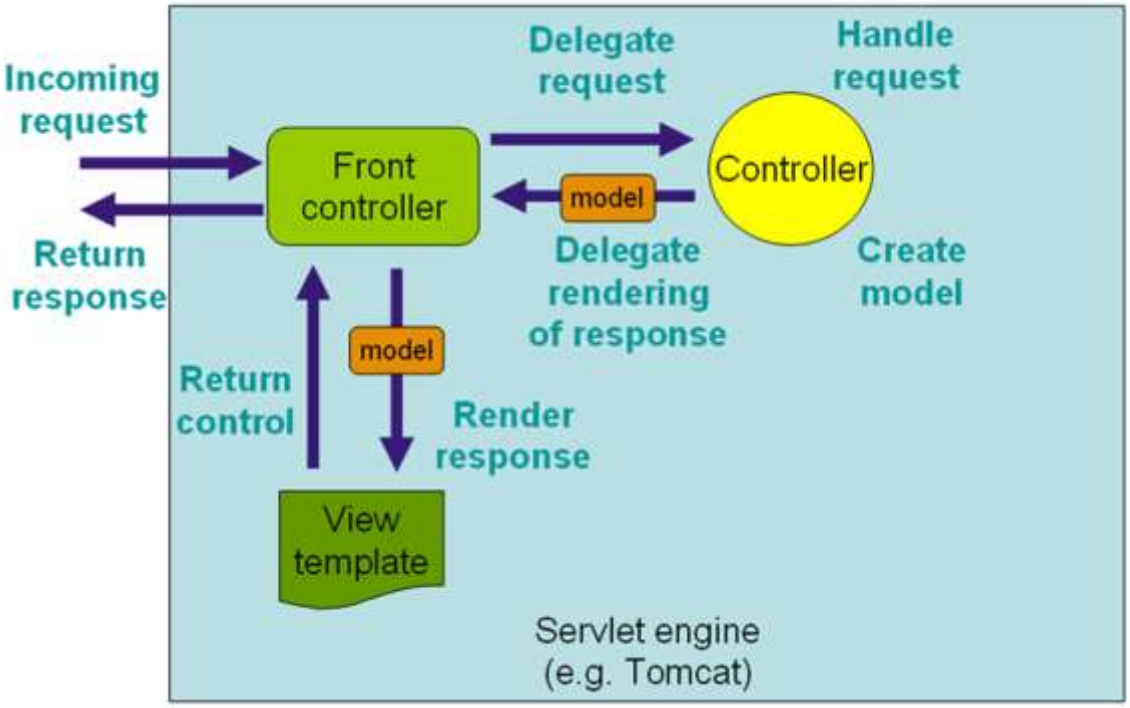


Рисунок 2.11 – Взаємодія компонентів Spring MVC

2.2.1 Структура бази даних

Структура бази даних описана в таблиці 2.1, схема бази даних зображена на рисунку 2.12.

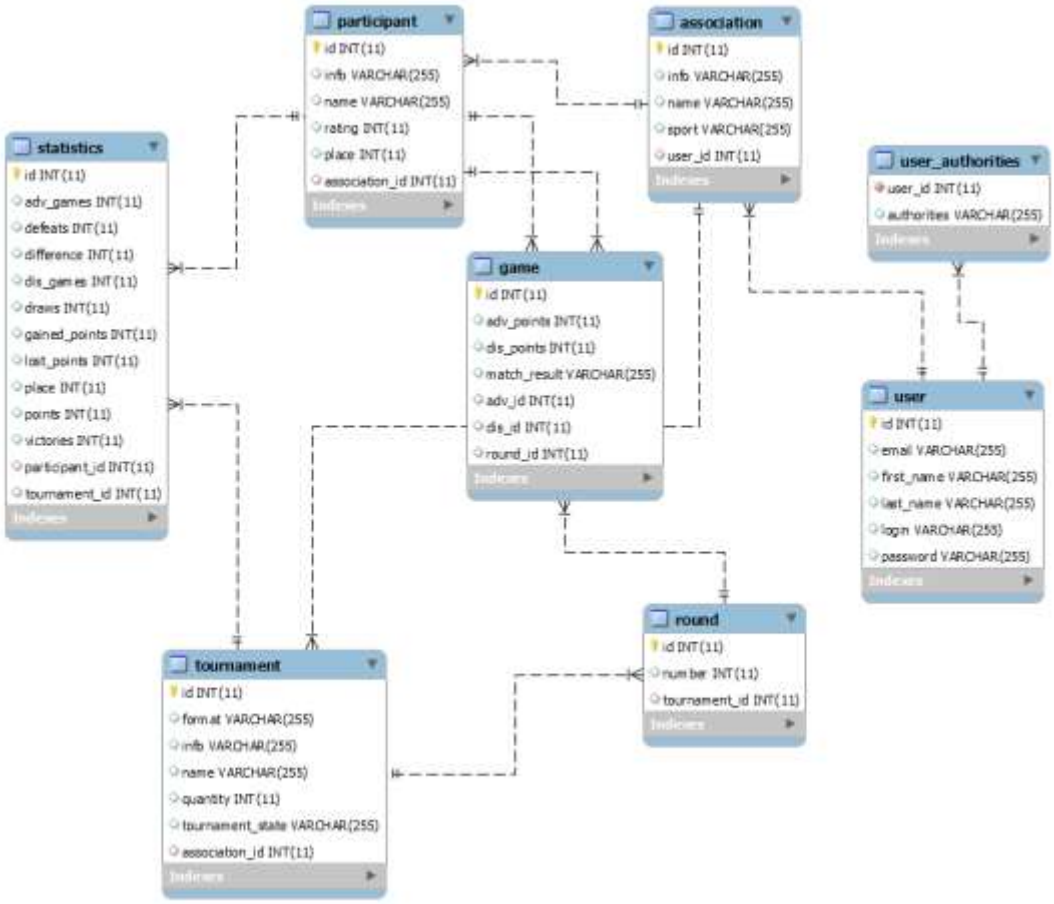


Рисунок 2.12 – Схема бази даних

Таблиця 2.1 – Опис таблиць бази даних

Назва таблиці	Опис таблиці
associations	Зберігає інформацію про асоціації.
games	Зберігає інформацію про матч.
participants	Зберігає інформацію про учасника/команду.
rounds	Зберігає інформацію про раунди турнірів.
statistics	Зберігає інформацію про становище команди/учасника у турнірі.
tournaments	Зберігає інформацію про турніри.

Продовження таблиці 2.1

users	Зберігає інформацію про користувачів.
user_authorities	Зберігає інформацію про роль користувача.

Далі детально розглянемо структуру кожної із таблиць бази даних.

Таблиця 2.2 – Опис таблиці associations

Колонка	Опис	Тип даних	Ключ
id	Ідентифікатор	int	PK
info	Інформація про асоціацію	varchar	
name	Назва асоціації	varchar	
sport	Назва дисципліни	varchar	
user_id	Ідентифікатор користувача, якому належить асоціація	int	FK

Таблиця 2.3 – Опис таблиці games

Колонка	Опис	Тип даних	Ключ
id	Ідентифікатор	int	PK
adv_points	Кількість очок, які було здобуто першим учасником/командою	int	
dis_points	Кількість очок, які було здобуто другим учасником/командою	int	
match_result	Стан матчу	varchar	

Продовження таблиці 2.3

adv_id	Ідентифікатор першого учасника/команди	int	FK
dis_id	Ідентифікатор другого учасника/команди	int	FK
round_id	Ідентифікатор раунду	int	FK

Таблиця 2.4 – Опис таблиці participants

Колонка	Опис	Тип даних	Ключ
id	Ідентифікатор	int	PK
name	Назва команди/ ім’я учасника	varchar	
rating	Кількість очок рейтингу	int	
place	Становище у таблиці рейтингу команд в асоціації	int	
association_id	Ідентифікатор асоціації	int	FK

Таблиця 2.5 – Опис таблиці rounds

Колонка	Опис	Тип даних	Ключ
id	Ідентифікатор	int	PK
number	Номер раунду	int	
tournament_id	Ідентифікатор турніру	int	FK

Таблиця 2.6 – Опис таблиці statistics

Колонка	Опис	Тип даних	Ключ
id	Ідентифікатор	int	PK
gained_points	Кількість набраних очок у матчах проти супротивників у матчах	varchar	
lost_points	Кількість очок, набраних супротивниками у матчах	int	
adv_games	Кількість ігор за сторону із перевагою	int	
dis_games	Кількість ігор за сторону без переваги	int	
victories	Кількість перемог	int	
draws	Кількість нічиїх	int	
defeats	Кількість поразок	int	
difference	Різниця між набутими очками учасника/команди та їх супротивників у матчах турніру	int	

Продовження таблиці 2.6

points	Кількість очок, які набираються за перемоги/нічиї	int	
place	Місце у турнірі	int	
participant_id	Ідентифікатор учасника	int	FK
tournament_id	Ідентифікатор турніру	int	FK

Таблиця 2.7 – Опис таблиці tournaments

Колонка	Опис	Тип даних	Ключ
id	Ідентифікатор	int	PK
format	Формат турніру	varchar	
info	Інформація про турнір	varchar	
name	Назва турніру	int	
quantity	Кількість команд/учасників у турнірі	int	
tournament_state	Стан турніру	varchar	
association_id	Ідентифікатор асоціації	int	FK

Таблиця 2.8 – Опис таблиці users

Колонка	Опис	Тип даних	Ключ
id	Ідентифікатор	int	PK

Продовження таблиці 2.8

email	Електронна пошта користувача	varchar	
first_name	Ім’я користувача	varchar	
last_name	Прізвище користувача	varchar	
login	Логін користувача	varchar	
password	Пароль користувача у зашифрованому вигляді	varchar	

2.3 Конструювання програмного забезпечення

Вже було зазначено, що даний додаток буде розроблений за патерном MVC. При розробці програмного забезпечення із використанням фреймворку Spring MVC є загальноприйняте розбиття структури класів на шари: сутностей, репозиторіїв, сервісів та контролерів.

Шар сутностей – складається із класів, які відповідно описують сутності. Структурна схема класів сутностей наведена на рисунку 2.13.

Рисунок 2.13 – Схема структурна класів сутностей

Шар репозиторії – це інтерфейси які розширюють інтерфейси, які надаються Spring Data. Вони відповідають за доступ до даних в базі даних. Репозиторії надають ряд базових методів та можливість створювати нові методи, які дозволяють не писати SQL-запити власноруч, використовуючи стандартизовані шаблони. Структурна схема класів репозиторіїв наведена на рисунку 2.14.

Рисунок 2.14 – Схема структурна класів репозиторіїв

Шар сервісів – це набір інтерфейсів та їх реалізації, в яких виконується вся бізнес логіка, яка виконується в програмі. Вони дістають дані використовуючи репозиторії. Структурна схема класів шару сервісів наведена в документі КПІ.ІП-6316.045440.07-99.СС Схема структурна діаграма класів сервісів.

Призначення кожного з сервісів описано у таблиці 2.9 – Призначення сервісів.

Таблиця 2.9 – Призначення сервісів

Назва сервісу	Призначення
AssociationService	Даний сервіс відповідає за роботу з асоціаціями, а саме: створення асоціацій, пошук асоціацій, отримання асоціацій.
GameService	Даний сервіс відповідає за обробку матчів: зберігає та дістає результати.
KnockoutService	Цей сервіс займається обробкою турнірів формату нокаут системи: проведення жеребкування, запуск турніру, завершення турніру, завершення раунду.
ParticipantService	Цей сервіс відповідає за роботу із командами/учасниками: їх створення, додавання в турнір, підрахунок рейтингу.

Продовження таблиці 2.9

RobinService	Цей сервіс займається обробкою турнірів формату кругова системи: проведення жеребкування, запуск турніру, завершення турніру, завершення раунду.
StatisticsService	Цей клас займається обробкою полів статистики.
SwissService	Цей сервіс займається обробкою турнірів формату швейцарської системи: проведення жеребкування, запуск турніру, завершення турніру, завершення раунду.
UserService	Сервіс, що відповідає за реєстрацію та автентифікацію користувача.
TournamentService	Цей сервіс є сервісом, який містить загальні методи, які займаються конвертуванням даних із DTO в об’єкти сутності та делегує проведення турніру класам, які реалізують різні формати.

Опис методів сервісів описаний у таблицях нижче.

Таблиця 2.10 – Опис сервіса AssociationService

Метод	Вхідні дані	Вихідні дані	Опис
createAssociation	Асоціація	Ідентифікатор асоціації	Створення асоціації
getAssociationsByUser	Користувач	Список Асоціацій	Знаходження списку асоціацій за користувачем

Продовження таблиці 2.10

findById	Ідентифікатор асоціації	Асоціація	Знаходження асоціації за ідентифікатором
search	Строка значення	Список асоціацій	Пошук асоціацій за ключовим словом
isOwner	Ідентифікатор асоціації	Бульове значення	Перевіряє чи є користувач власником організації

Таблиця 2.11 – Опис сервіса GameService

Метод	Вхідні дані	Вихідні дані	Опис
saveGameResult	Матч	-	Збереження матчу
getGameResult	Результат матчу	Кільсть очок, які набрав перший супротивник; кількість очок, які набрав другий супротивник	Встановлення результату матчу

Таблиця 2.12 – Опис сервіса ParticipantService

Метод	Вхідні дані	Вихідні дані	Опис

Продовження таблиці 2.12

getParticipantsByAssociation	Учасники	Список учасників	Отримання всіх учасників в асоціації у відсортованому порядку
getParticipantDTOByAssociation	Ідентифікатор асоціації	Список ДТО учасників	Отримання ДТО учасників зі списку учасників
ParticipantsToDTO	Учасники	ДТО учасників	Конвертування учасників в ДТО учасників
setStanding	Ідентифікатор асоціації	-	Встановлення положення учасників
changeRatingAfterGame	Ідентифікатор матчу	-	Підрахунок рейтингу учасників після матчу
calculateRating	Рейтинг гравця 1, рейтинг гравця 2, результат матчу	-	Обчислення зміни рейтингу за системою Ело
createParticipant	Учасник	-	Створення учасника

Таблиця 2.13 – Опис сервіса StatisticsService

Метод	Вхідні дані	Вихідні дані	Опис
createStatistics	Статистика	-	Створення статистики
updateStatisticsAfterGame	Статистика, матч	-	Оновлення статистики
updateAdvSideStatistics	Статистика, матч	-	Оновлення статистики учасника1
updateDisSideStatistics	Статистика, матч	-	Оновлення статистики гравця 2
getStatisticsDTOByTournament	Список статистики	Список ДТО статистики	Отримання дто статистики за турніром
convertStatisticsToDTO	Список статистики	Список ДТО статистики	Конвертування статистики в ДТО статистики

Таблиця 2.14 – Опис сервіса UserService

Метод	Вхідні дані	Вихідні дані	Опис
createUser	ДТО реєстрації	-	Створення користувача
isDuplicate	Строка	Бульове значення	Перевірка логіну на унікальність
getUser	ДТО автентифікації	Користувач	Пошук користувача

Таблиця 2.15 – Опис сервіса TournamentService

Метод	Вхідні дані	Вихідні дані	Опис
createTournament	ДТО турніру	Ідентифікатор турніру	Створення турніру
getTournamentsByAssociation	Ідентифікатор асоціації	Список турнірів	Отримання списку турнірів в асоціації
setParticipantsQuantity	Ідентифікатор турніру	-	Встановлення кількості учасників
getTournamentDTOById	Ідентифікатор турніру	Турнір	Отримання ДТО учасників по ідентифікатору
tournamentStart	Ідентифікатор турніру	-	Початок турнір
finishTournament	Ідентифікатор турніру	-	Завершення турніру
finishRound	Ідентифікатор турніру	-	Завершення раунду
getCurrentRound	Ідентифікатор турніру	Список ДТО матчів	Отримання поточного раунду
isLastRound	Ідентифікатор турніру	Бульове значення	Перевірка останній раунд зараз чи ні
search	Строка значення	Список турнірів	Пошук турніру за ключовим словом

Сервіси RobinService, KnockoutService, SwissService реалізують сервіс TournamentService, тому основна логіка в них повторюється.

Шар контролерів – це структура класів, яка приймає запити користувача та відправляє отриману інформацію у сервіси, які виконують потрібну бізнес логіку. Потім контролери відправляють відповідь користувачу. Структурна схема класів контролерів наведена на рисунку 2.15.

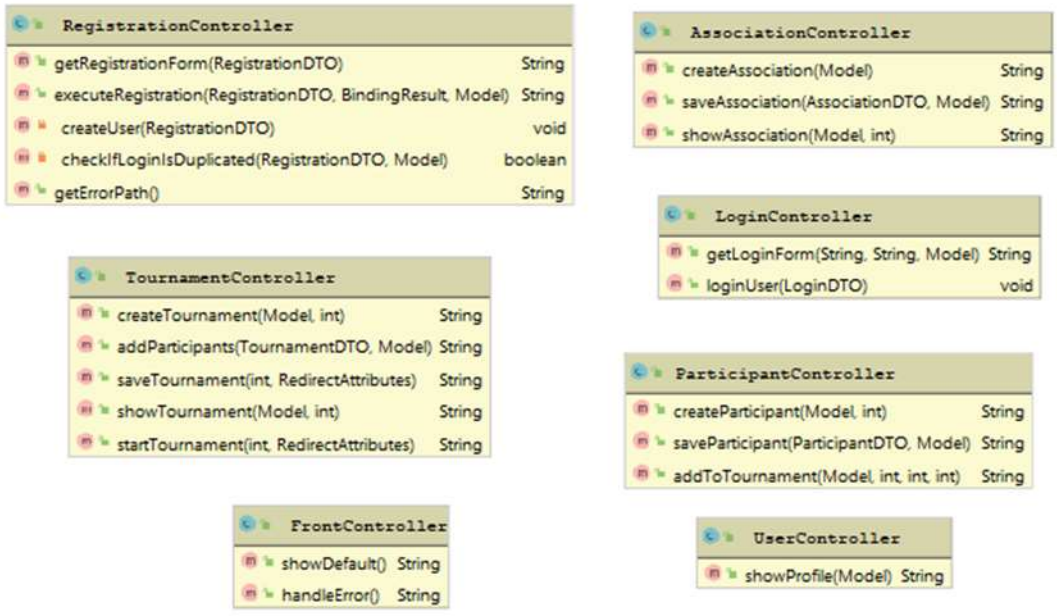


Рисунок 2.15 – Схема структурна класів контролерів

2.4 Аналіз безпеки даних

В даному додатку використовується фреймворк Spring Security. Він передбачає захист доступу до даних від несанкціонованого доступу, тобто відповідає за механізми автентифікації та авторизації. Кожен запит користувача проходить процес авторизації і в разі відсутності прав на доступ до даних, система видає помилку.

Паролі у базі даних не зберігається в звичайному вигляді. Для збереження паролів використовуються хешування паролів за допомогою алгоритму bcrypt. bcrypt був вибраний так як він є одним з алгоритмів, рекомендованих OWASP,

для хешування паролів.[8]

Bcrypt — адаптивна криптографічна функція формування ключа, що використовується для безпечного зберігання паролів. Розробники: Нільс Провос і David Mazières. Функція заснована на шифрі Blowfish, вперше представлена на USENIX у 1999 році. Для захисту від атак за допомогою райдужних таблиць bcrypt використовує сіль (salt); крім того, функція є адаптивною, час її роботи легко настроюється і її можна сповільнити, щоб ускладнити атаки перебором.[9]

2.5 Висновки по розділу

У даному розділі було проаналізовано бізнес-процеси інформаційної системи та проілюстровано з використанням діаграм BPMN. Було розроблену схему бази даних та описано шари класів ПЗ. Аналіз безпеки даних показав, що спроектована система має достатній рівень захисту.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості програмного забезпечення

Тестування є одним з найважливіших етапів розробки ПЗ.

Існує багато видів тестування, що використовують для серверів.

Модульне тестування (англ. Unit testing) — це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестованою.[10]

Інтеграційне тестування (англ. integration testing) — це фаза тестування програмного забезпечення, під час якої окремі модулі програми комбінуються та тестуються разом, у взаємодії. Інтеграційне тестування виконується після модульного тестування.[11]

Статичний аналіз коду (англ. static code analysis) — аналіз програмного забезпечення, який здійснюють (на відміну від динамічного аналізу) без реального виконання програм, що досліджуються.[12]

Перед розгортанням серверу у робочому середовищі, необхідно провести всі види тестів та статичний аналіз коду. Після закінчення тестів потрібно сформувати звіт про тестування, що включає в себе:

- список дефектів;
- оцінку якості програмного продукту;
- опис критичних дефектів;
- висновок про подальшу роботу: розгортання в робочому середовищі, чи повернення продукту на доопрацювання.

3.2 Опис процесів тестування

3.2.1 Документація функціональних тестів

Для контролю якості ПЗ було розроблено тест-кейси, перераховані в

таблиці 3.1.

Таблиця 3.1 – Опис функціональних тестів

Ідентифікатор	Опис
CF-1	Перевірити авторизацію користувача із коректними даними
CF-2	Перевірити відсутність можливості авторизації користувача з некоректними даними.
CF-3	Перевірити реєстрацію користувача, логін якого є унікальним
CF-4	Перевірити відсутність можливості реєстрації користувача із логіном, який вже існує в системі
CF-5	Перевірити можливість створення асоціації авторизованим користувачем
CF-6	Перевірити можливість створення турніру в асоціації авторизованим користувачем
CF-7	Перевірити можливість створення учасника в асоціації авторизованим користувачем
CF-8	Перевірити можливість додавати команду до турніру
CF-9	Перевірити неможливість створення команди в асоціації неавторизованим користувачем
CF-10	Перевірити неможливість створення турніру в асоціації неавторизованим користувачем

Продовження таблиці 3.1

CF-11	Перевірити можливість розпочати свій турнір авторизованому користувачу
CF-12	Перевірити відсутність можливості розпочати турнір неавторизованому користувачу
CF-13	Перевірити можливість заносити результати раунду свого турніру авторизованому користувачу
CF-14	Перевірити неможливість заносити результати раунду турніру неавторизованому користувачу
CF-15	Перевірити можливість пошуку користувача за його логіном
CF-16	Перевірити можливість пошуку асоціації за назвою
CF-17	Перевірити можливість пошуку турніру за назвою
CF-18	Перевірити коректність жеребкування турніру формату «Нокаут»
CF-19	Перевірити коректність жеребкування турніру формату «Круговий»
CF-20	Перевірити коректність жеребкування турніру формату «Швейцарська система»
CF-21	Перевірка коректності підрахунку зміни рейтингу

3.2.2 Документація функціональних тестів

У таблиці 3.2 наведено звіт про результати тестування, що включає в себе всі тест кейси, описані в таблиці 3.1.

Таблиця 3.2 – Звіт про результати тестування

Ідентифікатор тесту	Очікуваний результат	Фактичний результат	Статус тесту
CF-1	Користувач може авторизуватись ввівши коректні дані	Користувач може авторизуватись ввівши коректні дані	Пройдено
CF-2	Користувач не може авторизуватись ввівши коректні дані	Користувач не може авторизуватись ввівши коректні дані	Пройдено
CF-3	Користувач може зареєструватись вказавши унікальний логін	Користувач може зареєструватись вказавши унікальний логін	Пройдено
CF-4	Користувач не може зареєструватись вказавши неунікальний логін	Користувач не може зареєструватись вказавши неунікальний логін	Пройдено

Продовження таблиці 3.2

CF-5	Авторизований користувач може створити асоціацію	Авторизований користувач може створити асоціацію	Пройдено
CF-6	Авторизований користувач може створити турнір	Авторизований користувач може створити турнір	Пройдено
CF-7	Авторизований користувач може створити учасника в своїй асоціації	Авторизований користувач може створити учасника в своїй асоціації	Пройдено
CF-8	Авторизований користувач може при створенні турніру додавати команди із поточної асоціації	Авторизований користувач може при створенні турніру додавати команди із поточної асоціації	Пройдено
CF-9	Користувач не може створити учасника не в своїй асоціації	Користувач не може створити учасника не в своїй асоціації	Пройдено
CF-10	Користувач не може створити турнір не в своїй асоціації	Користувач не може створити турнір не в своїй асоціації	Пройдено

Продовження таблиці 3.2

CF-11	Авторизований користувач може розпочати свій турнір	Авторизований користувач може розпочати свій турнір	Пройдено
CF-12	Користувач не може розпочати не свій турнір	Користувач не може розпочати не свій турнір	Пройдено
CF-13	Авторизований користувач може заносити результати раунду свого турніру	Авторизований користувач може заносити результати раунду свого турніру	Пройдено
CF-14	Користувач не може заносити результати раунду не свого турніру	Користувач не може заносити результати раунду не свого турніру	Пройдено
CF-15	Користувач може шукати користувачів за логіном	Користувач може шукати користувачів за логіном	Пройдено
CF-16	Користувач може шукати турніри за назвою	Користувач може шукати турніри за назвою	Пройдено
CF-17	Користувач може шукати асоціації за назвою	Користувач може шукати турніри за назвою	Пройдено

Продовження таблиці 3.2

CF-18	Жеребкування турніру формату «Нокаут» відбувається правильно	Жеребкування турніру формату «Нокаут» відбувається правильно	Пройдено
CF-19	Жеребкування турніру формату «Круговий» відбувається правильно	Жеребкування турніру формату «Круговий» відбувається правильно	Пройдено
CF-20	Жеребкування турніру формату «Швейцарська система» відбувається правильно	Жеребкування турніру формату «Швейцарська система» відбувається правильно	Пройдено
CF-21	Підрахунок зміни рейтингу відбувається правильно	Підрахунок зміни рейтингу відбувається правильно	Пройдено

3.3 Знайдені недоліки та висновки

Було проведено аналіз якості програмного забезпечення, а саме серверного застосування. Було визначено найважливіші етапи в тестуванні серверних застосувань. Це модульне й інтеграційне тестування та статичний аналіз коду.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Діаграма розгортання компонентів застосунку представлена на рисунку 4.1.

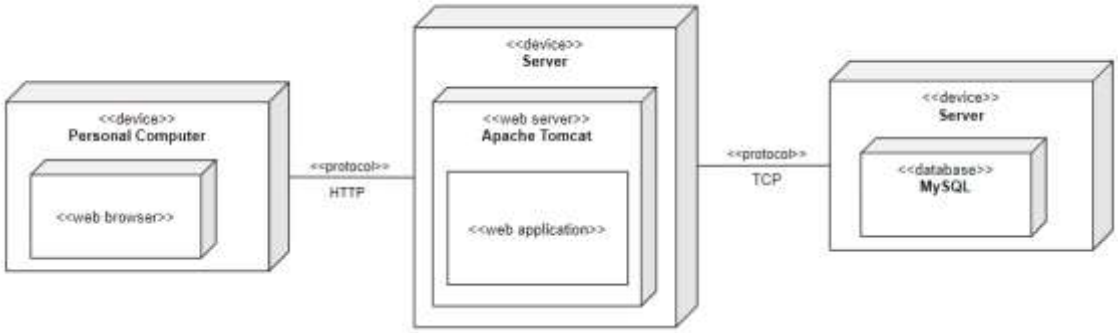


Рисунок 4.1 – Схема структурна розгортання програмного забезпечення

Серверна частина повинна бути розгорнута на сервері на якій встановлено одну з наступних ОС:

– Ubuntu 14.04 або вище;
– Windows Server 2008 R2 SP1 або вище;
– Windows 7 SP1 або вище;
– Red Hat Enterprise Linux 7 або вище.

Програмне забезпечення потребує встановлення наступних інструментів:

– JRE 11 або вище;
– MySQL 8.

Розгортання відбувається шляхом виконання JAR-файлу. Він містить у собі веб-сервер Apache Tomcat.

Оскільки дане ПЗ є веб-додатком, то клієнт має мати встановлений браузер на своєму девайсі.

4.2 Робота з програмним забезпеченням

Керівництво користувача описане у додатку Д «Керівництво користувача».

ВИСНОВКИ

Під час розробки дипломного було розглянуто предметну область турнірів та рейтингових систем, було проаналізовано існуючі аналоги та технічні рішення, які застосовуються для автоматизації організації турнірів.

Під час розгляду рейтингових систем вибір пав на рейтингову систему Ело у базовому вигляді для підрахунку змін рейтингу команд/учасників. Було проаналізовано та обрано наступні формати : швейцарська система, нокаут система, кругова система. Для кожного з них було обрано оптимальні алгоритми жеребкування. Програмне забезпечення було спроектовано із можливістю легко додавати нові формати у майбутньому.

Також було проведено аналіз якості і тестування даного програмного забезпечення.

Була написана інструкція розгортання серверної частини програмного забезпечення. Інструкції користувача описано в додатках.

Розроблене веб-застосування можна застосовувати різноманітним дитячим школам, аматорським і напів-професійним організаціям та іншим закладам, які займаються розвитком спорту або ігор будь-яких дисциплін, які проводять турніри із матчами у яких приймають участь дві конкуруючі сторони. Також даний веб-додаток зручний для швидкого створення турніру серед друзів чи інших груп людей.

ПЕРЕЛІК ПОСИЛАНЬ

1) Фреймворк [Електронний ресурс]: (Стаття) /Wikipedia – Електрон. дан. (1 файл). – 2020 – Режим доступу: https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA. – Назва з екрана.

2) Mac OS - [Електронний ресурс]: (Стаття) / Wikipedia – Електрон. дан. (1 файл). – 2020 – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C%D0%B2%D0%B8%D0%B4%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5%D1%80. – Назва з екрана.

3) Role of tournaments - [Електронний ресурс]: (Стаття) / Deafinitelyinclusive – Електрон. дан. (1 файл). – 2020 – Режим доступу: http://deafinitelyinclusive.co.uk/why-are-sports-tournaments-important/. – Назва з екрана.

4) Турнір [Електронний ресурс]: (Стаття) /Wikipedia – Електрон. дан. (1 файл). – 2020 – Режим доступу: https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%BE%D1%80%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D1%82%D1%83%D1%80%D0%BD%D0%B8%D1%80. – Назва з екрана.

5) Рейтингова система [Електронний ресурс]: (Стаття) /Wikipedia – Електрон. дан. (1 файл). – 2020 – Режим доступу: https://ru.qwe.wiki/wiki/Sports_rating_system. – Назва з екрана.

6) Spring Framework [Електронний ресурс]: (Стаття) /Tutorialspoint – Електрон. дан. (1 файл). – 2020 – Режим доступу: https://www.tutorialspoint.com/spring/spring_overview.htm. – Назва з екрана.

7) Hibernate [Електронний ресурс]: (Стаття) /GeeksForGeeks – Електрон. дан. (1 файл). – 2020 – Режим доступу: https://www.geeksforgeeks.org/introduction-to-hibernate-framework/. – Назва з екрана.

8) Password Storage Cheat Sheet [Електронний ресурс]: (Стаття) /OWASP – Електрон. дан. (1 файл). – 2018 – Режим доступу: https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet#Use_a_cryptographically_strong_credential-specific_salt. – Назва з екрана.

9) Bcrypt [Електронний ресурс]: (Стаття) / Wikipedia – Електрон. дан. (1 файл). – 2018 – Режим доступу: https://uk.wikipedia.org/wiki/Bcrypt. – Назва з екрана.

10) Модульне тестування [Електронний ресурс]: (Стаття) / Wikipedia – Електрон. дан. (1 файл). – 2018 – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F. – Назва з екрана.

11) Інтеграційне тестування [Електронний ресурс]: (Стаття) / Wikipedia – Електрон. дан. (1 файл). – 2018 – Режим доступу: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F. – Назва з екрана.

12) Статичний аналіз коду [Електронний ресурс]: (Стаття) / Wikipedia – Електрон. дан. (1 файл). – 2018 – Режим доступу: https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D1%96%D0%B7_%D0%BA%D0%BE%D0%B4%D1%83. – Назва з екрана.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ВЕБ-ЗАСТОСУВАННЯ ДЛЯ ОРГАНІЗАЦІЇ РЕЙТИНГОВИХ ТУРНІРІВ З ЖЕРЕБКУВАННЯМ ТА ДВОМА КОНКУРУЮЧИМИ СТОРОНАМИ

Технічне завдання

КПІ.ІП-6316.045440.02.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Д.О. Нечай

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____Є.П. Кошовець

Київ – 2020 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-застосування для організації рейтингових турнірів з жеребкуванням та двома конкуруючими сторонами

Галузь застосування: Уся спортивна та ігрова сфера, де можна проводити змагання у матчах якої беруть участі 2 конкуруючі команди

Наведене технічне завдання поширюється на розробку веб-застосування для організації рейтингових турнірів з жеребкуванням та двома конкуруючими сторонами, котра використовується для автоматизація процесу організації турнірів(тобто створення, проведення жеребкування, підрахунок зміни рейтингу та положення команд у турнірі) для дисциплін у яких в одній грі приймає участь 2 команди або учасника.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки веб-застосування для організації рейтингових турнірів з жеребкуванням та двома конкуруючими сторонами є завдання на дипломне проєктування, затверджене кафедрою автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім.Ігоря Сікорського).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для організації і проведення рейтингових турнірів із жеребкуванням, які проводять матчі, де приймає участь 2 конкуруючі сторони.

Метою створення розробки є автоматизація процесу організації турнірів(тобто створення, проведення жеребкування, підрахунок зміни рейтингу та положення команд у турнірі).

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1 Для користувача:

– реєстрація в системі;
– авторизація в системі;
– створення асоціації;
– створення турніру;
– створення команди/учасника в асоціацію;
– перегляд власного профілю;
– перегляд власної асоціації;
– перегляд власного турніру;
– пошук організатора;
– пошук асоціації;
– пошук турніру;
– перегляд профілів;
– перегляд асоціацій;
– перегляд турнірів;
– початок турніру;
– закінчення раунду;
– закінчення турніру.

4.1.1.2 Для системи:

– підрахунок зміни рейтингу;
– жеребкування раунду.

4.1.2 Розробку виконати на платформі Windows, але серверну частину веб застосунку можна розгортати на платформі Linux, за умови що на ній будуть встановлені усі передбачені програми та залежності.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації.

4.2.2 Передбачити захист від некоректних дій користувача.

4.2.3 Забезпечити цілісність інформації в базі даних.

4.2.4 Забезпечити постійний доступ до мережі інтернет.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96.

Не висуваються.

4.3.2 Обслуговування та обслуговуючий персонал.

Не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

4.4.2 Мінімальна конфігурація технічних засобів:

4.4.2.1 Тип процесору

Процесори Intel i3 або кращі.

4.4.2.2 Об‘єм ОЗП

512 Мб, або більше.

4.5 Вимоги до інформаційної та програмної сумісності

– Програмне забезпечення повинно працювати під управлінням

операційних систем сімейства WIN64(Windows 7, Windows 8, Windows 10) або Unix;

– Вхідні дані повинні бути представлені в наступному форматі: HTTP-запит до сервера;

– Результати повинні бути представлені в наступному форматі: HTTP-відповідь з сервера;

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати установочну версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

а) Супроводжувальна документація:

1) пояснювальна записка;

2) керівництво користувача;

3) програма та методика тестування.

б) Довідникова документація:

1) Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні мати всі необхідні коментарі;

2) програмне забезпечення повинно мати вбудовану довідку.

в) Графічна документація:

1) схема структурна варіантів використань;

2) схема структурна класів прошарку сервісів;

3) креслення вигляду екранних форм.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проєкту	15.03.2020	
2.	Розробка технічного завдання	25.03.2020	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	29.03.2020	Специфікації програмного забезпечення
4.	Проєктування структури програмного забезпечення, проєктування компонентів	20.04.2020	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	30.04.2020	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05.2020	Тести, результати тестування
7.	Розробка матеріалів текстової частини проєкту	24.05.2020	Пояснювальна записка.
8.	Розробка матеріалів графічної частини проєкту	17.05.2020	Графічний матеріал проєкту
9.	Оформлення технічної документації проєкту	24.05.2020	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

7.1 Види випробувань

Тестування розробленого програмного продукту виконується відповідно до «Програми та методики тестування».

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ВЕБ-ЗАСТОСУВАННЯ ДЛЯ ОРГАНІЗАЦІЇ РЕЙТИНГОВИХ ТУРНІРІВ З ЖЕРЕБКУВАННЯМ ТА ДВОМА КОНКУРУЮЧИМИ СТОРОНАМИ

Опис програми

КПІ.ІП-6316.045440.03.13

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Д.О. Нечай

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____Є.П. Кошовець

Київ – 2020 року

Тексти програмного коду

Веб-застосування для організації рейтингових турнірів з жеребкуванням та двома конкуруючими сторонами

(Найменування програми (документа))

DVD-R

(Вид носія даних)

44 арк., 93 КБ

(Обсяг програми (документа) , арк.,) Кб)

Київ - 2020

```
@Controller
public class AssociationController {

    private AssociationService associationService;
    private TournamentService tournamentService;
    private ParticipantService participantService;
    private UserService userService;

    @Autowired
    public AssociationController(AssociationService
associationService, ParticipantService participantService,
                                 TournamentService
tournamentService, UserService userService) {
        this.associationService = associationService;
        this.tournamentService = tournamentService;
        this.participantService = participantService;
        this.userService = userService;
    }

    @GetMapping("/createAssociation")
    public String createAssociation(Model model){
        model.addAttribute("association", new AssociationDTO());
        return "create_association";
    }


    @PostMapping("/saveAssociation")
    public String saveAssociation(AssociationDTO dto,
RedirectAttributes ra){
        int associationId =
associationService.createAssociation(dto);
        ra.addAttribute("associationId",
String.valueOf(associationId));
        return "redirect:/association";
    }


    @RequestMapping("/association")
    public String showAssociation(Model model,
@RequestParam("associationId") int id){
        Association curAssociation =
associationService.findById(id);
        model.addAttribute("association_id",id);
        //model.addAttribute("association", curAssociation);
        model.addAttribute("tournaments",
tournamentService.getTournamentsByAssociation(id));
        //return "association/query?id=" + associationId;
        model.addAttribute("isOwner",
associationService.isOwner(id));
        return "association";
    }
```

```
    @PostMapping("/redirectToAssociation")
    public String saveTournament(@RequestParam("associationId")
int associationId, RedirectAttributes ra){
        ra.addAttribute("associationId",
String.valueOf(associationId));
        return "redirect:/association";
    }

    @RequestMapping("/redirectToCreateAssociation")
    public String redirectToCreateAssociation(){
        return "redirect:/createAssociation";
    }

    @RequestMapping("/showStanding")
    private String showStanding(@RequestParam("associationId") int
associationId, Model model){
        participantService.setPlaces(associationId);
        model.addAttribute("participants",
participantService.getParticipantsByAssociationOrdered(association
Id));
        return "standing";
    }

    @PostMapping("/redirectShowStanding")
    public String
redirectShowStanding(@RequestParam("associationId") int
associationId, RedirectAttributes ra){
        ra.addAttribute("associationId",
String.valueOf(associationId));
        return "redirect:/showStanding";
    }

}

@Controller
public class FrontController {

    @GetMapping("/")
    public String showDefault() {
        return "redirect:/profile";
    }

    @GetMapping("/error")
    public String handleError() {

        return "error";
    }
}
```

```
@Controller
public class GameController {
    private AssociationService associationService;
    private ParticipantService participantService;
    private TournamentService tournamentService;
    private StatisticsService statisticsService;
    private GameService gameService;
    @Autowired
    public GameController(AssociationService associationService,
ParticipantService participantService,
                          TournamentService tournamentService,
StatisticsService statisticsService,
                          GameService gameService) {
        this.associationService = associationService;
        this.participantService = participantService;
        this.tournamentService = tournamentService;
        this.statisticsService = statisticsService;
        this.gameService = gameService;
    }

    @PostMapping("/addGameResult")
    public String addToTournament(RedirectAttributes ra,
@RequestParam("gameId") int gameId,
                                  @RequestParam("tournamentId")
int tournamentId,

@ModelAttribute("result_game")GameDTO dto){
        gameService.saveGameResult(dto, gameId);
        statisticsService.updateStatisticsAfterGame(gameId,
tournamentId);
        participantService.changeRatingAfterGame(gameId);
        ra.addAttribute("tournamentId",
String.valueOf(tournamentId));
        return "redirect:/tournament";
    }
}

@Controller
@RequestMapping("/login")
public class LoginController {

    private final UserService userService;

    @Autowired
    public LoginController(UserService userService) {
        this.userService = userService;
    }

    @GetMapping
    public String getLoginForm(@RequestParam(value = "error",
required = false) String error,
```

```
                                    @RequestParam(value = "logout",
required = false) String logout,
                                    Model model) {
        model.addAttribute("error", error != null);
        model.addAttribute("logout", logout != null);
        return "login";
    }

    @ResponseStatus(HttpStatus.ACCEPTED)
    @PostMapping
    public void loginUser(LoginDTO dto)throws CredentialsException
{
        userService.getUser(dto);
    }

}

@Controller
public class ParticipantController {
    private AssociationService associationService;
    private ParticipantService participantService;
    private TournamentService tournamentService;
    private StatisticsService statisticsService;

    @Autowired
    public ParticipantController(AssociationService
associationService, ParticipantService participantService,
                                 TournamentService
tournamentService, StatisticsService statisticsService) {
        this.associationService = associationService;
        this.participantService = participantService;
        this.tournamentService = tournamentService;
        this.statisticsService = statisticsService;
    }

    @PostMapping("/createParticipant")
    public String createParticipant(Model model, @RequestParam int
id){
        model.addAttribute("participant",new ParticipantDTO(id));
        return "create_participant";
    }


    @PostMapping("/saveParticipant")
    public String saveParticipant(ParticipantDTO dto,
RedirectAttributes ra){
        participantService.createParticipant(dto);
        ra.addAttribute("associationId",
String.valueOf(dto.getAssociationId()));
        return "redirect:/association";
    }
```

```
    @PostMapping("/addToTournament")
    public String addToTournament(Model model,
@RequestParam("tournamentId") int tournamentId,
                                  @RequestParam("participantId") int
participantId,
                                    @RequestParam("associationId")
int associationId){

        statisticsService.createStatistics(tournamentId,
participantId);
        model.addAttribute("tournament_id", tournamentId);
        model.addAttribute("participants",
participantService.getParticipantsDTOByAssociation(associationId,
tournamentId));
        model.addAttribute("association_id", associationId);

        return "add_participants";
    }

    @PostMapping("/addToTournamentTest")
    public String addToTournamentTest(Model model,
@RequestParam("tournamentId") int tournamentId,
                                    @RequestParam("participantId")
int participantId,
                                    @RequestParam("associationId")
int associationId){

        statisticsService.createStatistics(tournamentId,
participantId);
        model.addAttribute("tournament_id", tournamentId);
        model.addAttribute("participants",
participantService.getParticipantsDTOByAssociation(associationId,
tournamentId));
        model.addAttribute("association_id", associationId);

        return "add_participants";
    }
}

@Log4j2
@Controller
@RequestMapping("/registration")
public class RegistrationController implements ErrorController {

    private UserService userService;
    private MessageSource messageSource;

    @Autowired
    public RegistrationController(UserService userService,
MessageSource messageSource) {
```

```
        this.userService = userService;
        this.messageSource = messageSource;
    }

    @GetMapping
    public String getRegistrationForm(@ModelAttribute("user")
RegistrationDTO dto){
        return "registration";
    }

    @ResponseStatus(HttpStatus.CREATED)
    @PostMapping
    public String executeRegistration(@ModelAttribute("user")
@Valid RegistrationDTO dto,
                                      BindingResult bindingResult,
Model model){

        if (checkIfLoginIsDuplicated(dto, model)) return
"registration";
        if (bindingResult.hasErrors()) return "registration";
        createUser(dto);
        return "login";
    }

    private void createUser(@ModelAttribute("user") @Valid
RegistrationDTO dto) {
         userService.createUser(dto);
         log.info("Created user");


    }

    private boolean
checkIfLoginIsDuplicated(@ModelAttribute("user") @Valid
RegistrationDTO dto, Model model) {
        if(userService.isDuplicate(dto.getLogin())){
            model.addAttribute("error_message",
messageSource.getMessage("Not unique login",
                    null,
                    LocaleContextHolder.getLocale()) +
dto.getLogin());
            return true;
        }
        return false;
    }

    @Override
    public String getErrorPath() {
        return "/error";
    }
```

```
}


@Controller
public class SearchController {
    private TournamentService tournamentService;
    private AssociationService associationService;

    @Autowired
    public SearchController(TournamentService tournamentService,
AssociationService associationService) {
        this.tournamentService = tournamentService;
        this.associationService = associationService;
    }

    @GetMapping("/tournamentSearch")
    public String tournamentSearch(Model model){
        model.addAttribute("tournaments",
tournamentService.search(""));
        model.addAttribute("search_item", new SearchDTO());
        return "tournament_search";
    }
    @PostMapping("/tournamentSearch")
    public String tournamentSearch(SearchDTO dto, Model model){
        model.addAttribute("tournaments",
tournamentService.search(dto.getValue()));
        model.addAttribute("search_item", dto);
        return "tournament_search";
    }

    @GetMapping("/associationSearch")
    public String associationSearch(Model model){
        model.addAttribute("associations",
associationService.search(""));
        model.addAttribute("search_item", new SearchDTO());
        return "association_search";
    }
    @PostMapping("/associationSearch")
    public String associationSearch(SearchDTO dto, Model model){
        model.addAttribute("associations",
associationService.search(dto.getValue()));
        model.addAttribute("search_item", dto);
        return "association_search";
    }
}


@Controller
public class TournamentController {
```

```
    private TournamentService tournamentService;
    private ParticipantService participantService;
    private AssociationService associationService;
    private StatisticsService statisticsService;

    @Autowired
    public TournamentController(TournamentService
tournamentService, ParticipantService participantService,
                                StatisticsService
statisticsService, AssociationService associationService) {
        this.tournamentService = tournamentService;
        this.participantService = participantService;
        this.associationService = associationService;
        this.statisticsService = statisticsService;
    }

    @PostMapping("/createTournament")
    public String createTournament(Model model,@RequestParam int
id){
        model.addAttribute("tournament", new
TournamentDTO(id,"UPCOMING"));
        model.addAttribute("all_formats", Format.values());
        return "create_tournament";
    }

    @ResponseStatus(HttpStatus.CREATED)
    @PostMapping("/addParticipants")
    public String addParticipants(@ModelAttribute("tournament")
TournamentDTO dto,
                                  Model model){
        int tournamentId =
tournamentService.createTournament(dto);
        participantService.setPlaces(dto.getAssociationId());
        model.addAttribute("tournament_id", tournamentId);
        model.addAttribute("participants", participantService

.getParticipantsDTOByAssociation(dto.getAssociationId(),
tournamentId));
        model.addAttribute("association_id",
dto.getAssociationId());

        return "add_participants";
    }

    @RequestMapping("/saveTournament")
    public String saveTournament(@RequestParam("tournamentId") int
tournamentId, RedirectAttributes ra){
        tournamentService.setParticipantQuantity(tournamentId);
        ra.addAttribute("tournamentId",
String.valueOf(tournamentId));
        return "redirect:/tournament";
```

```
    }

    @RequestMapping("/tournament")
    public String showTournament(Model model,
@RequestParam("tournamentId") int tournamentId){
        model.addAttribute("tournament_id",tournamentId);
        model.addAttribute("current_round",
tournamentService.getCurrentRound(tournamentId));
        model.addAttribute("tournament",
tournamentService.getTournamentDTOById(tournamentId));
        model.addAttribute("statistics",
statisticsService.getStatisticsDTOByTournament(tournamentId));
        model.addAttribute("is_last_round",
tournamentService.isLastRound(tournamentId));
        model.addAttribute("result_game",new GameDTO());
        model.addAttribute("isOwner",
tournamentService.isOwner(tournamentId));
        return "tournament";
    }

    @RequestMapping("/startTournament")
    public String startTournament(@RequestParam("tournamentId")
int tournamentId, RedirectAttributes ra)
    {
        tournamentService.tournamentStart(tournamentId);
        ra.addAttribute("tournamentId",
String.valueOf(tournamentId));
        return "redirect:/tournament";
    }

    @PostMapping("/finishRound")
    public String finishRound(@RequestParam("tournamentId") int
tournamentId, RedirectAttributes ra){
        tournamentService.finishRound(tournamentId);
        ra.addAttribute("tournamentId",
String.valueOf(tournamentId));
        return "redirect:/tournament";
    }

    @PostMapping("/finishTournament")
    public String finishTournament(@RequestParam("tournamentId")
int tournamentId, RedirectAttributes ra){
        tournamentService.finishTournament(tournamentId);
        ra.addAttribute("tournamentId",
String.valueOf(tournamentId));
        return "redirect:/tournament";
    }

    @PostMapping("/createTournamentTest")
    public String createTournamentTest(Model model,@RequestParam
int id){
```

```
        model.addAttribute("tournament", new
TournamentCreationDTO(id,"UPCOMING"));
        model.addAttribute("participantsInAssociation",
participantService.getParticipantsDTOByAssociation(id));
        model.addAttribute("all_formats", Format.values());
        return "create_tournament_test";
    }

    @RequestMapping("/saveTournamentTest")
    public String saveTournamentTest(@ModelAttribute("tournament")
TournamentCreationDTO dto , RedirectAttributes ra){
        int tournamentId =
tournamentService.createTournament(dto);
        statisticsService.createStatistics(tournamentId,
dto.getParticipants());
        tournamentService.setParticipantQuantity(tournamentId);
        ra.addAttribute("tournamentId",
String.valueOf(tournamentId));
        return "redirect:/tournament";
    }
}

@Controller
public class UserController {

    private AssociationService associationService;
    private TournamentService tournamentService;

    @Autowired
    public UserController(AssociationService associationService,
TournamentService tournamentService) {
        this.associationService = associationService;
        this.tournamentService = tournamentService;
    }

    @GetMapping("/profile")
    public String showProfile(Model model){
        model.addAttribute("associations",
associationService.getAssociationsByUser());

        return "profile";
    }


}


@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
```

```
@Entity
@EqualsAndHashCode
@Table(name = "association")
public class Association {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Column(name = "name")
    private String name;

    @Column(name = "sport")
    private String sport;

    @Column(name = "info")
    private String info;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @ManyToOne(cascade= {CascadeType.PERSIST, CascadeType.MERGE,
            CascadeType.DETACH, CascadeType.REFRESH})
    @JoinColumn(name="user_id")
    private User user;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToMany(mappedBy="association",
            cascade= {CascadeType.PERSIST, CascadeType.MERGE,
                    CascadeType.DETACH, CascadeType.REFRESH})
    List<Tournament> tournaments;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToMany(mappedBy="association",
            cascade= {CascadeType.PERSIST, CascadeType.MERGE,
                    CascadeType.DETACH, CascadeType.REFRESH})
    List<Participant> participants;
}

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
@EqualsAndHashCode
@Table(name = "game")
public class Game {

    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Enumerated(EnumType.STRING)
    @Column(name = "match_result")
    private MatchResult matchResult;

    @Column(name = "adv_points")
    private int advPoints;

    @Column(name = "dis_points")
    private int disPoints;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @ManyToOne(cascade= {CascadeType.PERSIST, CascadeType.MERGE,
            CascadeType.DETACH, CascadeType.REFRESH})
    @JoinColumn(name="round_id")
    private Round round;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @ManyToOne(cascade= {CascadeType.PERSIST, CascadeType.MERGE,
            CascadeType.DETACH, CascadeType.REFRESH})
    @JoinColumn(name="adv_id")
    private Participant advSide;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @ManyToOne(cascade= {CascadeType.PERSIST, CascadeType.MERGE,
            CascadeType.DETACH, CascadeType.REFRESH})
    @JoinColumn(name="dis_id")
    private Participant disSide;
}
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
@EqualsAndHashCode
@Table(name = "participant")
public class Participant {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Column(name = "name")
    private String name;
```

```
    @Column(name = "info")
    private String info;

    @Column(name = "rating")
    private int rating;

    @Column(name = "place")
    private int standing;


    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @ManyToOne(cascade= {CascadeType.PERSIST, CascadeType.MERGE,
            CascadeType.DETACH, CascadeType.REFRESH})
    @JoinColumn(name="association_id")
    private Association association;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToMany(mappedBy="participant",
            cascade= {CascadeType.PERSIST, CascadeType.MERGE,
                    CascadeType.DETACH, CascadeType.REFRESH})
    List<Statistics> statistics;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToMany(mappedBy="advSide",
            cascade= {CascadeType.PERSIST, CascadeType.MERGE,
                    CascadeType.DETACH, CascadeType.REFRESH})
    List<Game> advGames;

    @OneToMany(mappedBy="disSide",
            cascade= {CascadeType.PERSIST, CascadeType.MERGE,
                    CascadeType.DETACH, CascadeType.REFRESH})
    List<Game> disGames;
}


@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
@EqualsAndHashCode
@Table(name = "round")
public class Round {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;
```

```
    @Column(name = "number")
    private int number;


    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @ManyToOne(cascade= {CascadeType.PERSIST, CascadeType.MERGE,
            CascadeType.DETACH, CascadeType.REFRESH})
    @JoinColumn(name="tournament_id")
    private Tournament tournament;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToMany(mappedBy="round",
            cascade= {CascadeType.PERSIST, CascadeType.MERGE,
                    CascadeType.DETACH, CascadeType.REFRESH})
    List<Game> games;
}


@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
@EqualsAndHashCode
@Table(name = "statistics")
public class Statistics {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Column(name = "adv_games")
    private int advGames;

    @Column(name = "dis_games")
    private int disGames;

    @Column(name = "victories")
    private int victories;

    @Column(name = "defeats")
    private int defeats;

    @Column(name = "draws")
    private int draws;

    @Column(name = "gained_points")
```

```
    private int gainedPoints;

    @Column(name = "lost_points")
    private int lostPoints;

    @Column(name = "difference")
    private int difference;

    @Column(name = "points")
    private int points;

    @Column(name = "place")
    private int place;


    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @ManyToOne(cascade= {CascadeType.PERSIST, CascadeType.MERGE,
            CascadeType.DETACH, CascadeType.REFRESH})
    @JoinColumn(name="tournament_id")
    private Tournament tournament;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @ManyToOne(cascade= {CascadeType.PERSIST, CascadeType.MERGE,
            CascadeType.DETACH, CascadeType.REFRESH})
    @JoinColumn(name="participant_id")
    private Participant participant;


}


@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
@EqualsAndHashCode
@Table(name = "tournament")
public class Tournament {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Column(name = "name")
    private String name;

    @Column(name = "quantity")
```

```
    private int quantity;

    @Column(name = "info")
    private String info;

    @Enumerated(EnumType.STRING)
    @Column(name = "tournament_state")
    private TournamentState tournamentState;

    @Enumerated(EnumType.STRING)
    @Column(name = "format")
    private Format format;


    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @ManyToOne(cascade= {CascadeType.PERSIST, CascadeType.MERGE,
            CascadeType.DETACH, CascadeType.REFRESH})
    @JoinColumn(name="association_id")
    private Association association;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToMany(mappedBy="tournament",
            cascade= {CascadeType.PERSIST, CascadeType.MERGE,
                    CascadeType.DETACH, CascadeType.REFRESH})
    List<Statistics> statistics;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToMany(mappedBy="tournament",
            cascade= {CascadeType.PERSIST, CascadeType.MERGE,
                    CascadeType.DETACH, CascadeType.REFRESH})
    List<Round> rounds;

}


@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Entity
@EqualsAndHashCode
@Table(name = "user", uniqueConstraints =
{@UniqueConstraint(columnNames = {"login"})})
public class User implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
```

```
    private int id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "login", unique = true)
    private String login;

    @Column(name = "password")
    private String password;

    @Column(name = "email")
    private String email;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @ElementCollection(targetClass = Role.class, fetch =
FetchType.EAGER)
    @CollectionTable(name = "user_authorities", joinColumns =
@JoinColumn(name = "user_id"))
    @Enumerated(EnumType.STRING)
    private Set<Role> authorities;

    @ToString.Exclude
    @EqualsAndHashCode.Exclude
    @OneToMany(mappedBy="user",
               cascade= {CascadeType.PERSIST, CascadeType.MERGE,
                    CascadeType.DETACH, CascadeType.REFRESH})
    private List<Association> associations;

    @Override
    public String getUsername() {
        return getLogin();
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
```

```
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}


@Repository
public interface AssociationRepository extends
JpaRepository<Association,Long > {
    List<Association> findByUser(User user);
    Association findById(int id);
    Association findByUserAndName(User user, String name);
    List<Association> findAllByNameContaining(String value);
    Association findByTournaments(Tournament tournament);
}

@Repository
public interface GameRepository extends JpaRepository<Game,Long >
{
    List<Game> findAllByRoundOrderById(Round round);

    Game findById(int id);
}

@Repository
public interface ParticipantRepository extends
JpaRepository<Participant,Long> {
    List<Participant> findByAssociation(Association association);
    Participant findById(int id);
    List<Participant>
findAllByAssociationOrderByRatingDesc(Association association);
    List<Participant>
findAllByAssociationOrderByStandingAsc(Association association);

}

@Repository
public interface RoundRepository extends JpaRepository<Round,Long>
{
    Round findByTournamentAndNumber(Tournament tournament, int
number);
    List<Round> findAllByTournament(Tournament tournament);
}

@Repository
public interface StatisticsRepository extends
JpaRepository<Statistics,Long> {
    List<Statistics> findAllByTournament(Tournament tournament);
```

```
    List<Statistics>
findAllByTournamentOrderByPointsDescDifferenceDescGainedPointsDesc
(Tournament tournament);
    Statistics findByTournamentAndParticipant(Tournament
tournament, Participant participant);
}


@Repository
public interface TournamentRepository extends
JpaRepository<Tournament,Long> {
    List<Tournament> findByAssociation(Association association);
    Tournament findByNameAndAssociation(String name, Association
association);
    Tournament findById(int id);
    List<Tournament> findAllByNameContaining(String name);
}


@Repository
public interface UserRepository extends JpaRepository<User,Long> {
    User findByLogin(String login);
    User findByLoginAndPassword(String login, String password);
    User findByAssociations(Association association);
}


@Service
public class AssociationService {

    private final UserRepository userRepository;
    private final AssociationRepository associationRepository;
    private final TournamentRepository tournamentRepository;

    @Autowired
    public AssociationService(UserRepository userRepository,
AssociationRepository associationRepository,
                              TournamentRepository
tournamentRepository) {
        this.userRepository = userRepository;
        this.associationRepository = associationRepository;
        this.tournamentRepository = tournamentRepository;
    }

    public int createAssociation(AssociationDTO dto){
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        User user =
userRepository.findByLogin(authentication.getName());
        Association associationToAdd = Association.builder()
                .user(user)
```

```
                .info(dto.getInfo())
                .sport(dto.getSport())
                .name(dto.getName())
                .build();
        associationRepository.save(associationToAdd);
        return associationRepository.findByUserAndName(user,
dto.getName()).getId();
    }

    public List<Association> getAssociationsByUser(){
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        User user =
userRepository.findByLogin(authentication.getName());
        return associationRepository.findByUser(user);
    }

    public Association findById(int id){
        return associationRepository.findById(id);
    }
    public List<Association> search(String value){
        if(value.equals("")) return
associationRepository.findAll();
        return
associationRepository.findAllByNameContaining(value);
    }

    public boolean isOwner(int associationId){
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        User user =
userRepository.findByLogin(authentication.getName());
        User owner =
userRepository.findByAssociations(associationRepository
                .findById(associationId));
        return user.equals(owner);
    }
}


@Service
public class GameService{

    private final AssociationRepository associationRepository;
    private final TournamentRepository tournamentRepository;
    private final ParticipantRepository participantRepository;
    private final StatisticsRepository statisticsRepository;
    private final GameRepository gameRepository;
    private final RoundRepository roundRepository;

    @Autowired
```

```
    public GameService(AssociationRepository
associationRepository,
                       TournamentRepository tournamentRepository,
                       ParticipantRepository
participantRepository,
                       StatisticsRepository statisticsRepository,
                       GameRepository gameRepository,
                       RoundRepository roundRepository
    ) {
        this.participantRepository = participantRepository;
        this.associationRepository = associationRepository;
        this.tournamentRepository = tournamentRepository;
        this.statisticsRepository = statisticsRepository;
        this.gameRepository = gameRepository;
        this.roundRepository = roundRepository;
    }

    public void saveGameResult(GameDTO dto, int gameId){
        Game game = gameRepository.findById(gameId);
        game.setAdvPoints(dto.getAdvPoints());
        game.setDisPoints(dto.getDisPoints());
        game.setMatchResult(getMatchResult(dto.getAdvPoints(),
dto.getDisPoints()));
        gameRepository.save(game);
    }

    private MatchResult getMatchResult(int advPoints, int
disPoints){
        if(advPoints> disPoints)
            return MatchResult.VICTORY;
        else if(advPoints < disPoints)
            return MatchResult.DEFEAT;
        else return MatchResult.DRAW;
    }
}

@Service
public class KnockoutService {
    public static final int SECOND_ROUND = 2;
    private final StatisticsRepository statisticsRepository;
    private final GameRepository gameRepository;
    private final RoundRepository roundRepository;

    @Autowired
    public KnockoutService(StatisticsRepository
statisticsRepository, GameRepository gameRepository,
RoundRepository roundRepository) {
        this.statisticsRepository = statisticsRepository;
        this.gameRepository = gameRepository;
        this.roundRepository = roundRepository;
    }
```

```
    public void start(Tournament tournament){
        Round round = createRound(1, tournament);
        List<Participant> participants =
makeStartDraw(tournament);
        createGamesForRound(participants.size(),round,
participants);
    }

    public void makeRound(Tournament tournament){
        int roundNumber = tournament.getRounds().size()+1;
        Round round = createRound(roundNumber,tournament);
        List<Participant> participants = drawRound(tournament);
        createGamesForRound(participants.size() ,round,
participants);
    }

    public List<Game> getCurrentRound(Round round){
        List<Game> games =
gameRepository.findAllByRoundOrderById(round);
        return games;
    }


    private Round createRound(int roundNumber, Tournament
tournament){
        Round roundToAdd = Round.builder()
                .number(roundNumber)
                .tournament(tournament)
                .build();
        roundRepository.save(roundToAdd);
        return roundToAdd;
    }

    private void createGamesForRound(int quantity, Round round,
final List<Participant> participants){
        Random index = new Random();
        while(participants.size()!= 0){
            Participant firstParticipant =
participants.get(index.nextInt(participants.size()));
            participants.remove(firstParticipant);
            Participant secondParticipant =
participants.get(index.nextInt(participants.size()));
            participants.remove(secondParticipant);
           createGame(firstParticipant,secondParticipant, round);
        }
    }
    private void createGame(Participant advSide, Participant
disSide, Round round){
        Game gameToAdd = Game.builder()
                .advSide(advSide)
```

```
                .disSide(disSide)
                .round(round)
                .matchResult(MatchResult.NOTPLAYED)
                .build();
        gameRepository.save(gameToAdd);
    }


    private List<Participant> makeStartDraw(Tournament
tournament){
        List<Statistics> statistics =
statisticsRepository.findAllByTournament(tournament);
        List<Participant> participants = new ArrayList<>();
        for(Statistics s: statistics){
            participants.add(s.getParticipant());
        }
        int quantity = participants.size();
        boolean isAdditionalRound = calculateLog2(quantity) !=
Math.floor(calculateLog2(quantity));
        if(isAdditionalRound){

participants.sort(Comparator.comparing(Participant::getRating));
            Collections.reverse(participants);
            int extra = quantity -
(int)Math.pow(2,Math.floor(calculateLog2(quantity)));
            while(participants.size()!=2*extra){
                participants.remove(0);
            }
        }
        return participants;
    }
    private List<Participant> drawRound(Tournament tournament){
        List<Statistics> statistics;
        List<Participant> participants = new ArrayList<>();
        int roundNumber = tournament.getRounds().size()+1;
        List<Game> games =
gameRepository.findAllByRoundOrderById(roundRepository
                .findByTournamentAndNumber(tournament,
roundNumber-1));
          for(Game game : games){
              if(game.getMatchResult() == MatchResult.VICTORY)
                  participants.add(game.getAdvSide());
               else if(game.getMatchResult() ==
MatchResult.DEFEAT)
                  participants.add(game.getDisSide());
          }

        if(roundNumber == SECOND_ROUND){
            statistics = statisticsRepository
                    .findAllByTournament(tournament);
            for(Statistics s : statistics){
```

```
                if(s.getVictories()==0 && s.getDefeats()==0)
                    participants.add(s.getParticipant());
            }
        }
        return participants;
    }

    public boolean isLastRound(Tournament tournament) {
        int roundNumber = tournament.getRounds().size();
        int quantity = tournament.getQuantity();
        int lastRound = (int)Math.ceil(calculateLog2(quantity));
        return roundNumber == lastRound;
    }

    private double calculateLog2(int value){
        return (Math.log(value) / Math.log(2));
    }
}

@Service
public class ParticipantService{
    private final AssociationRepository associationRepository;
    private final ParticipantRepository participantRepository;
    private final StatisticsRepository statisticsRepository;
    private final TournamentRepository tournamentRepository;
    private final GameRepository gameRepository;

    @Autowired
    public ParticipantService(AssociationRepository
associationRepository, StatisticsRepository statisticsRepository,
                              ParticipantRepository
participantRepository, TournamentRepository tournamentRepository,
                              GameRepository gameRepository) {
        this.associationRepository = associationRepository;
        this.participantRepository = participantRepository;
        this.statisticsRepository = statisticsRepository;
        this.tournamentRepository = tournamentRepository;
        this.gameRepository = gameRepository;
    }

    public List<Participant>
getParticipantsByAssociation(Association association){
        return
participantRepository.findByAssociation(association);
    }

    public List<Participant>
getParticipantsByAssociationOrdered(int associationId){
        return
participantRepository.findAllByAssociationOrderByStandingAsc(
                associationRepository.findById(associationId));
```

```
    }

    public List<ParticipantAddingDTO>
getParticipantsDTOByAssociation(int associationId, int
tournamentId){
        List<Statistics> statistics =
getAddedParticipants(tournamentId);
        List<Participant> participants =
participantRepository.findAllByAssociationOrderByStandingAsc(
                associationRepository.findById(associationId));
        for(Statistics s : statistics){
            participants.remove(s.getParticipant());
        }
        return participantsToDTO(participants);
    }

    public List<ParticipantAddingDTO>
getParticipantsDTOByAssociation(int associationId) {
        List<Participant> participants =
participantRepository.findAllByAssociationOrderByStandingAsc(
                associationRepository.findById(associationId));
        return participantsToDTO(participants);
    }

    public List<ParticipantAddingDTO>
participantsToDTO(List<Participant> participants ){
        List<ParticipantAddingDTO> dtoList = new ArrayList<>();
        for(Participant participant : participants){
            dtoList.add(new
ParticipantAddingDTO(participant.getName(),participant.getRating()
,participant.getStanding(),
            participant.getId()));
        }
        return  dtoList;
    }

    public void createParticipant(ParticipantDTO dto){
        Association association =
associationRepository.findById(dto.getAssociationId());
        Participant participantToAdd= Participant.builder()
                .association(association)
                .info(dto.getInfo())
                .rating(dto.getRating())
                .name(dto.getName())
                .build();

        participantRepository.save(participantToAdd);
    }

    public List<Statistics> getAddedParticipants(int
tournamentId){
```

```
        return
statisticsRepository.findAllByTournament(tournamentRepository.find
ById(tournamentId));

    }

    public void setStanding(int associationId){
        List<Participant> participants =
participantRepository.findAllByAssociationOrderByRatingDesc
                (associationRepository.findById(associationId));
        for (int i = 0, place = 1; i < participants.size(); i++,
place++) {
            participants.get(i).setStanding(place);
        }
    }

    public void changeRatingAfterGame(int gameId){
        Game game = gameRepository.findById(gameId);
        Participant advSide = game.getAdvSide();
        Participant disSide = game.getDisSide();
        int advRating = advSide.getRating();
        int disRating = disSide.getRating();

advSide.setRating(calculateRating(advRating,disRating,game.getMatc
hResult()));

disSide.setRating(calculateRating(disRating,advRating,game.getMatc
hResult()));
        participantRepository.save(advSide);
        participantRepository.save(disSide);
    }

    private int calculateRating(int advRating, int disRating ,
MatchResult matchResult){
        double E = 1. / (1 + Math.pow(10,(disRating-
advRating)/400.));
        double S=0;
        switch(matchResult){
            case VICTORY:
                S=1;
                break;
            case DRAW:
                S=0.5;
                break;
            case DEFEAT:
                S=0;
                break;
        }
        return (int)(advRating + 16 * (S-E));
    }
```

```
    public void setPlaces(int associationId){
        List<Participant> participants = participantRepository.
                findAllByAssociationOrderByRatingDesc(

(associationRepository.findById(associationId)));
        for (int i = 0, place = 1; i < participants.size(); i++,
place++) {
            participants.get(i).setStanding(place);
            participantRepository.save(participants.get(i));
        }
    }
}

@Service
public class RobinService{

    private final StatisticsRepository statisticsRepository;
    private final GameRepository gameRepository;
    private final RoundRepository roundRepository;

    @Autowired
    public RobinService(StatisticsRepository statisticsRepository,
GameRepository gameRepository, RoundRepository roundRepository) {
        this.statisticsRepository = statisticsRepository;
        this.gameRepository = gameRepository;
        this.roundRepository = roundRepository;
    }

    public void start(Tournament tournament){
        int quantity = tournament.getQuantity() ;
        Round round = createRound(1, tournament);
        List<Participant> participants =
makeStartDraw(tournament);
        createGamesForRound(quantity,round, participants);
    }

    public void makeRound(Tournament tournament){
        int quantity = tournament.getQuantity() ;
        int roundNumber = tournament.getRounds().size()+1;
        Round round = createRound(roundNumber,tournament);
        List<Participant> participants =
drawRound(roundNumber,getStartDraw(tournament, quantity));
        createGamesForRound(quantity ,round, participants);
    }

    public List<Game> getCurrentRound(Round round){
        List<Game> games =
gameRepository.findAllByRoundOrderById(round);
        for(Game game : games){
            if(game.getAdvSide().equals(game.getDisSide())){
                games.remove(game);
```

```
                break;
            }
        }
        return games;
    }

    public boolean isLastRound(Tournament tournament){
        int roundNumber = tournament.getRounds().size();
        int quantity = tournament.getQuantity();
        if(quantity %2 == 1) {
            return roundNumber == quantity;
        } else{
            return  roundNumber == quantity -1;
        }


    }

    private List<Participant> makeStartDraw(Tournament
tournament){

        List<Statistics> statistics =
statisticsRepository.findAllByTournament(tournament);
        List<Participant> participants = new ArrayList<>();
        for(Statistics s: statistics){
            participants.add(s.getParticipant());
        }
        return participants;
    }

    private List<Participant> getStartDraw(Tournament tournament,
int quantity){
        Round round =
roundRepository.findByTournamentAndNumber(tournament, 1);
        List<Game> games =
gameRepository.findAllByRoundOrderById(round);
        List<Participant> participants = new ArrayList<>();
        if(quantity % 2 == 1 ){
            participants.add(games.get(games.size()-
1).getAdvSide());
            games.remove(games.get(games.size()-1));
        }
        for(Game game : games){
            participants.add(game.getAdvSide());
        }
        for(Game game : games){
            participants.add(game.getDisSide());
        }

        return participants;
    }
```

```
    private void createGame(Participant advSide, Participant
disSide, Round round){
        Game gameToAdd = Game.builder()
                .advSide(advSide)
                .disSide(disSide)
                .round(round)
                .matchResult(MatchResult.NOTPLAYED)
                .build();
        gameRepository.save(gameToAdd);
    }

    private List<Participant> drawRound(int roundNumber, final
List<Participant> startParticipants){
        int quantity = startParticipants.size() ;
        List<Participant> curParticipants = new ArrayList<>();
        while(curParticipants.size() < quantity)
curParticipants.add(null);
        Participant firstParticipant = startParticipants.get(0);
        startParticipants.remove(0);
        for(int i = 0, newI = roundNumber -1; i< quantity -
roundNumber ; i++, newI ++) {
            Participant tempParticipant =
startParticipants.get(i);
            curParticipants.set(newI , tempParticipant);
        }
        for(int i = quantity - roundNumber, newI = 0; i< quantity
- 1; i++, newI++){
            Participant tempParticipant =
startParticipants.get(i);
            curParticipants.set(newI, tempParticipant);
        }
        curParticipants.add(0,firstParticipant);
        return  curParticipants;
    }

    private Round createRound(int roundNumber, Tournament
tournament){
        Round roundToAdd = Round.builder()
                .number(roundNumber)
                .tournament(tournament)
                .build();
        roundRepository.save(roundToAdd);
        return roundToAdd;
    }

    private void createGamesForRound(int quantity, Round round,
List<Participant> participants){
        int half = quantity / 2;
        for( int i = 0; i < half; i++){
```

```
createGame(participants.get(i),participants.get(i+half), round);
        }
        if(participants.size() % 2  == 1){
            createGame(participants.get(quantity-
1),participants.get(quantity-1), round);
        }
    }

}

@Service
public class StatisticsService{

    private final StatisticsRepository statisticsRepository;
    private final TournamentRepository tournamentRepository;
    private final ParticipantRepository participantRepository;
    private final GameRepository gameRepository;

    @Autowired
    public StatisticsService(StatisticsRepository
statisticsRepository,
                             TournamentRepository
tournamentRepository,
                             ParticipantRepository
participantRepository,
                             GameRepository gameRepository) {
        this.statisticsRepository = statisticsRepository;
        this.tournamentRepository = tournamentRepository;
        this.participantRepository = participantRepository;
        this.gameRepository = gameRepository;
    }

    public void createStatistics(int tournamentId, int
participantId) {
        Tournament tournament =
tournamentRepository.findById(tournamentId);
        Participant participant =
participantRepository.findById(participantId);
        Statistics statisticsToAdd = Statistics.builder()
                .participant(participant)
                .tournament(tournament)
                .defeats(0)
                .difference(0)
                .draws(0)
                .victories(0)
                .lostPoints(0)
                .gainedPoints(0)
                .advGames(0)
                .disGames(0)
                .place(0)
                .build();
```

```
            statisticsRepository.save(statisticsToAdd);
        }

        public List<StatisticsDTO> getStatisticsDTOByTournament(int
tournamentId) {
            List<Statistics> statistics = statisticsRepository.

findAllByTournamentOrderByPointsDescDifferenceDescGainedPointsDesc
(
                    (tournamentRepository.findById(tournamentId)));
            for (int i = 0, place = 1; i < statistics.size(); i++,
place++) {
                statistics.get(i).setPlace(place);
            }
            return convertStatisticsToDTO(statistics);
        }

        public List<StatisticsDTO>
convertStatisticsToDTO(List<Statistics> statistics) {
            List<StatisticsDTO> dto = new ArrayList<>();
            for (Statistics s : statistics) {
                Participant participant = s.getParticipant();
                dto.add(StatisticsDTO.builder()
                        .name(participant.getName())
                        .rating(participant.getRating())
                        .difference(s.getDifference())
                        .gainedPoints(s.getGainedPoints())
                        .lostPoints(s.getLostPoints())
                        .defeats(s.getDefeats())
                        .draws(s.getDraws())
                        .standing(s.getPlace())
                        .victories(s.getVictories())
                        .points(s.getPoints())
                        .build()
                );
            }
            return dto;
        }

        public void updateStatisticsAfterGame(int gameId, int
tournamentId){
            Game game = gameRepository.findById(gameId);
            Tournament tournament =
tournamentRepository.findById(tournamentId);
            Statistics advSideStatistics = statisticsRepository
                    .findByTournamentAndParticipant(tournament,
game.getAdvSide());
            Statistics disSideStatistics = statisticsRepository
                    .findByTournamentAndParticipant(tournament,
game.getDisSide());
            updateAdvSideStatistics(advSideStatistics, game);
```

```
        updateDisSideStatistics(disSideStatistics, game);
        statisticsRepository.save(advSideStatistics);
        statisticsRepository.save(disSideStatistics);

    }

    private void updateAdvSideStatistics(Statistics statistics,
Game game){
        statistics.setAdvGames(statistics.getAdvGames()+1);

statistics.setGainedPoints(statistics.getGainedPoints()+game.getAd
vPoints());

statistics.setLostPoints(statistics.getLostPoints()+game.getDisPoi
nts());
        statistics.setDifference(statistics.getGainedPoints()-
statistics.getLostPoints());
        switch(game.getMatchResult()){
            case VICTORY:
                statistics.setPoints(statistics.getPoints() + 3);
                statistics.setVictories(statistics.getVictories()
+ 1);
                break;
            case DRAW:
                statistics.setPoints(statistics.getPoints() + 1);
                statistics.setDraws(statistics.getDraws() + 1);
                break;
            case DEFEAT:
                statistics.setDefeats(statistics.getDefeats()+1);
                break;
        }
    }

    private void updateDisSideStatistics(Statistics statistics,
Game game){
        statistics.setDisGames(statistics.getDisGames()+1);

statistics.setGainedPoints(statistics.getGainedPoints()+game.getDi
sPoints());

statistics.setLostPoints(statistics.getLostPoints()+game.getAdvPoi
nts());
        statistics.setDifference(statistics.getGainedPoints()-
statistics.getLostPoints());
        switch(game.getMatchResult()){
            case DEFEAT:
                statistics.setPoints(statistics.getPoints() + 3);
                statistics.setVictories(statistics.getVictories()
+ 1);
                break;
            case DRAW:
```

```
                statistics.setPoints(statistics.getPoints() + 1);
                statistics.setDraws(statistics.getDraws() + 1);
                break;
            case VICTORY:
                statistics.setDefeats(statistics.getDefeats()+1);
                break;
        }
    }

    public void createStatistics(int tournamentId, String[]
participants) {
        for(String participant :participants){
            createStatistics(tournamentId,
Integer.parseInt(participant));
        }
    }
}


@Service
public class SwissService  {
    private final StatisticsRepository statisticsRepository;
    private final GameRepository gameRepository;
    private final RoundRepository roundRepository;

    @Autowired
    public SwissService(StatisticsRepository statisticsRepository,
GameRepository gameRepository, RoundRepository roundRepository) {
        this.statisticsRepository = statisticsRepository;
        this.gameRepository = gameRepository;
        this.roundRepository = roundRepository;
    }

    public void start(Tournament tournament){
        int quantity = tournament.getQuantity() ;
        Round round = createRound(1, tournament);
        List<Participant> participants =
makeStartDraw(tournament);
        createGamesForRound(quantity,round, participants);
    }

    public void makeRound(Tournament tournament){
        int quantity = tournament.getQuantity() ;
        int roundNumber = tournament.getRounds().size()+1;
        Round round = createRound(roundNumber,tournament);
        List<Participant> participants = drawRound(tournament,
quantity);
        createGamesForRound(quantity ,round, participants);
    }

    public List<Game> getCurrentRound(Round round){
```

```
        List<Game> games =
gameRepository.findAllByRoundOrderById(round);
        /*for(Game game : games){
            if(game.getAdvSide().equals(game.getDisSide())){
                games.remove(game);
                break;
            }
        }*/
        return games;
    }

    public boolean isLastRound(Tournament tournament) {
        int roundNumber = tournament.getRounds().size();
        int quantity = tournament.getQuantity();
        int lastRound = calculateLastRound(quantity);
        return roundNumber == lastRound;
    }
    private int calculateLastRound(int quantity) {
        return (int)Math.round(calculateLog2(quantity) +
calculateLog2(3*2));
    }
    private double calculateLog2(int value){
        return Math.log(value) / Math.log(2);
    }

    private Round createRound(int roundNumber, Tournament
tournament){
        Round roundToAdd = Round.builder()
                .number(roundNumber)
                .tournament(tournament)
                .build();
        roundRepository.save(roundToAdd);
        return roundToAdd;
    }

    private void createGamesForRound(int quantity, Round round,
List<Participant> participants){
        int half = quantity / 2;
        for( int i = 0; i < half; i++){

createGame(participants.get(i),participants.get(i+half), round);
        }
        if(participants.size() % 2  == 1){
            createGame(participants.get(quantity-
1),participants.get(quantity-1), round);
        }
    }

    private void createGame(Participant advSide, Participant
disSide, Round round){
        Game gameToAdd = Game.builder()
```

```
                .advSide(advSide)
                .disSide(disSide)
                .round(round)
                .matchResult(MatchResult.NOTPLAYED)
                .build();
        gameRepository.save(gameToAdd);
    }


    private List<Participant> makeStartDraw(Tournament
tournament){
        List<Statistics> statistics =
statisticsRepository.findAllByTournament(tournament);
        List<Participant> participants = new ArrayList<>();
        for(Statistics s: statistics){
            participants.add(s.getParticipant());
        }

participants.sort(Comparator.comparing(Participant::getRating));
        Collections.reverse(participants);
        return participants;
    }

    private List<Participant> drawRound(Tournament tournament, int
quantity){
        List<Round> rounds =
roundRepository.findAllByTournament(tournament);
        List<Statistics> statistics = statisticsRepository

.findAllByTournamentOrderByPointsDescDifferenceDescGainedPointsDes
c(tournament);
        List<Participant> participants = new ArrayList<>();
        List<Participant> draw = new ArrayList<>();
        List<Game> games = new ArrayList<>();

        for(Statistics s: statistics){
            participants.add(s.getParticipant());
        }

        for(Round round : rounds){

games.addAll(gameRepository.findAllByRoundOrderById(round));
        }

        while(draw.size()!=quantity) draw.add(null);

        if(quantity % 2 == 1 ){
            draw.set(quantity-1, participants.get(quantity-1));
            participants.remove(quantity-1);
        }
```

```
        for(int i = 0;i< quantity/2; i++){
            Participant firstParticipant = participants.get(0);
            participants.remove(firstParticipant);
            draw.set(i,firstParticipant);
            for(Participant secondParticipant : participants){
                if(!havePlayed(firstParticipant,secondParticipant,
games)){
                    draw.set(i+quantity/2, secondParticipant);
                    participants.remove(secondParticipant);
                    break;
                }
            }
        }
        return draw;
    }
    private boolean havePlayed(Participant first, Participant
second, List<Game> games){
        for(Game game :games){
            if((game.getAdvSide().equals(first) &&
game.getDisSide().equals(second))
                || (game.getAdvSide().equals(second) &&
game.getDisSide().equals(first)))
                return true;
        }
        return false;
    }
    private void createGameForOdd(Participant participant, Round
round){
        Game gameToAdd = Game.builder()
                            .advSide(participant)
                            .disSide(participant)
                            .advPoints(1)
                            .disPoints(0)
                            .round(round)
                            .matchResult(MatchResult.VICTORY)
                            .build();
        gameRepository.save(gameToAdd);
    }
}

@Service
public class TournamentService  {

    private final AssociationRepository associationRepository;
    private final TournamentRepository tournamentRepository;
    private final RoundRepository roundRepository;
    private final UserRepository userRepository;
    private final RobinService robinService;
    private final SwissService swissService;
    private final KnockoutService knockoutService;
```

```
    @Autowired
    public TournamentService(AssociationRepository
associationRepository,
                             TournamentRepository
tournamentRepository,
                             UserRepository userRepository,
                             RoundRepository roundRepository,
                             RobinService robinService,
                             SwissService swissService,
                             KnockoutService knockoutService
    ) {
        this.associationRepository = associationRepository;
        this.tournamentRepository = tournamentRepository;
        this.roundRepository = roundRepository;
        this.robinService = robinService;
        this.swissService = swissService;
        this.knockoutService = knockoutService;
        this.userRepository = userRepository;
    }


    public int createTournament(TournamentDTO tournament) {
        Association association =
associationRepository.findById(tournament.getAssociationId());
        Tournament tournamentToAdd = Tournament.builder()
                .association(association)
                .info(tournament.getInfo())
                .format(Format.valueOf(tournament.getFormat()))
                .name(tournament.getName())

.tournamentState(TournamentState.valueOf(tournament.getStatus()))
                .statistics(new ArrayList<>())
                .build();
        tournamentRepository.save(tournamentToAdd);
        return
tournamentRepository.findByNameAndAssociation(tournament.getName()
, association).getId();
    }

    public int createTournament(TournamentCreationDTO tournament)
{
        Association association =
associationRepository.findById(tournament.getAssociationId());
        Tournament tournamentToAdd = Tournament.builder()
                .association(association)
                .info(tournament.getInfo())
                .format(Format.valueOf(tournament.getFormat()))
                .name(tournament.getName())

.tournamentState(TournamentState.valueOf(tournament.getStatus()))
                .statistics(new ArrayList<>())
```

```
                .build();
        tournamentRepository.save(tournamentToAdd);
        return
tournamentRepository.findByNameAndAssociation(tournament.getName()
, association).getId();
    }

    public List<Tournament> getTournamentsByAssociation(int
associationId) {
        return
tournamentRepository.findByAssociation(associationRepository.findB
yId(associationId));

    }

    public void setParticipantQuantity(int tournamentId) {
        Tournament tournament =
tournamentRepository.findById(tournamentId);
        tournament.setQuantity(tournament.getStatistics().size());
        tournamentRepository.save(tournament);
    }

    public TournamentDTO getTournamentDTOById(int tournamentId) {
        Tournament tournament =
tournamentRepository.findById(tournamentId);
        return TournamentDTO.builder().name(tournament.getName())
                .format(tournament.getFormat().toString())
                .info(tournament.getInfo())

.status(tournament.getTournamentState().toString())

.associationId(tournament.getAssociation().getId())
                .build();
    }
    public void tournamentStart(int tournamentId){
        Tournament tournament =
tournamentRepository.findById(tournamentId);
        tournament.setTournamentState(TournamentState.ONGOING);
        tournamentRepository.save(tournament);
        switch(tournament.getFormat()){
            case PLAY_OFF:
                knockoutService.start(tournament);
                break;
            case SWISS:
                swissService.start(tournament);
                break;
            case ROUND_ROBIN:
                robinService.start(tournament);
                break;

        }
```

```
    }

    public void finishRound(int tournamentId){
        Tournament tournament =
tournamentRepository.findById(tournamentId);
        switch(tournament.getFormat()){
            case PLAY_OFF:
                knockoutService.makeRound(tournament);
                break;
            case SWISS:
                swissService.makeRound(tournament);
                break;
            case ROUND_ROBIN:
                robinService.makeRound(tournament);
                break;
            default:
        }
    }

    public void finishTournament(int tournamentId){
        Tournament tournament =
tournamentRepository.findById(tournamentId);
        tournament.setTournamentState(TournamentState.COMPLETED);
        tournamentRepository.save(tournament);
    }

    public List<GameDTO> getCurrentRound(int tournamentId){
        Tournament tournament =
tournamentRepository.findById(tournamentId);
        int roundNumber = tournament.getRounds().size();
        if(roundNumber == 0) {return null;}
        Round round =
roundRepository.findByTournamentAndNumber(tournament,
roundNumber);
        switch(tournament.getFormat()){
            case SWISS:
                return
convertGameToDTO(swissService.getCurrentRound(round));
            case ROUND_ROBIN:
                return
convertGameToDTO(robinService.getCurrentRound(round));
            case PLAY_OFF:
                return
convertGameToDTO(knockoutService.getCurrentRound(round));
                default:
                    return null;
        }
    }

    private List<GameDTO> convertGameToDTO(List<Game> games){
        List<GameDTO> dto = new ArrayList<>();
```

```
        for(Game game : games){
            GameDTO dtoItem = GameDTO.builder()
                    .advName(game.getAdvSide().getName())
                    .disName(game.getDisSide().getName())
                    .advPoints(game.getAdvPoints())
                    .disPoints(game.getDisPoints())
                    .id(game.getId())
                    .matchResult(game.getMatchResult().toString())
                    .build();
            if(dtoItem.getAdvName().equals(dtoItem.getDisName())){
                dtoItem.setDisName("");
            }
            dto.add(dtoItem);
        }
        return dto;
    }

    public boolean isLastRound(int tournamentId){
        Tournament tournament =
tournamentRepository.findById(tournamentId);

        switch (tournament.getFormat()){
            case ROUND_ROBIN:
                return robinService.isLastRound(tournament);
            case SWISS:
                return swissService.isLastRound(tournament);
            case PLAY_OFF:
                return  knockoutService.isLastRound(tournament);
        }
        return false;
    }

    public List<Tournament> search(String value){
        return
tournamentRepository.findAllByNameContaining(value);
    }

    public boolean isOwner(int tournamentId){
        Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
        User user =
userRepository.findByLogin(authentication.getName());
        User owner =
userRepository.findByAssociations(associationRepository

.findByTournaments(tournamentRepository.findById(tournamentId)));
        return owner.equals(user);
    }
}
```

```
@Slf4j
@Service
public class UserService implements UserDetailsService {

    private final UserRepository userRepository;
    private final AssociationRepository associationRepository;
    private final PasswordEncoder passwordEncoder;
    private final MessageSource messageSource;

    @Autowired
    public UserService(UserRepository userRepository
,AssociationRepository associationRepository, PasswordEncoder
passwordEncoder, MessageSource messageSource) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
        this.messageSource = messageSource;
        this.associationRepository = associationRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String login) throws
UsernameNotFoundException {
        User user = userRepository.findByLogin(login);

        if (user == null) {
            throw new UsernameNotFoundException(login);
        }

        return user;
    }

    public void createUser(RegistrationDTO dto) {

        try{
            User userToAdd = User.builder()
                    .firstName(dto.getFirstName())
                    .lastName(dto.getLastName())
                    .login(dto.getLogin())

.password(passwordEncoder.encode(dto.getPassword()))
                    .email(dto.getEmail())
                    .authorities(Collections.singleton(Role.USER))
                    .build();

            userRepository.save(userToAdd);
            log.info("User saved successfully!");

        }catch(DataIntegrityViolationException exception){
            log.warn("Login not unique: " + dto.getLogin());
            throw new
LoginNotUniqueException(messageSource.getMessage(
```

```
                    "Login not unique:",
                    null,
                    LocaleContextHolder.getLocale()) +
dto.getLogin(), exception);
        }
    }

    public boolean isDuplicate(String login){

        return userRepository.findByLogin(login) != null;
    }

    public User getUser(LoginDTO dto) throws CredentialsException
{
        User user =
userRepository.findByLoginAndPassword(dto.getLogin(),
dto.getPassword());

        if(Objects.isNull(user)) {
            log.warn(dto + " there is no such user record in
database");
            System.out.println( "there is no such user record in
database");
            throw new CredentialsException("Invalid credentials");
        }

        if (!dto.getPassword().equals(user.getPassword())) {
            log.warn(dto + " incorrect password");
            System.out.println( "incorrect password");
            throw new CredentialsException("Invalid credentials");
        }

        log.info(dto + " user successfully logged in");
        System.out.println( " user successfully logged in");
        return user;
    }



}
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ВЕБ-ЗАСТОСУВАННЯ ДЛЯ ОРГАНІЗАЦІЇ РЕЙТИНГОВИХ ТУРНІРІВ З ЖЕРЕБКУВАННЯМ ТА ДВОМА КОНКУРУЮЧИМИ СТОРОНАМИ

Програма та методика тестування

КПІ.ІП-6316.045440.04.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Д.О. Нечай

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____Є.П. Кошовець

Київ – 2020 року

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Web-застосування для організації рейтингових турнірів з жеребкуванням та двома конкуруючими сторонами, написаний мовою програмування Java та побудований на архітектурному шаблоні MVC.

2 МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

- функціональна працездатність команд та інтерактивних елементів;
- коректність роботи алгоритмів жеребкування та підрахунку рейтингу;
- забезпечення належного рівня безпеки даних;
- зручність роботи з веб-застосуванням;
- відповідність вимогам Технічного завдання.

3 МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

– функціональне тестування, зокрема на рівні Critical path test (базове тестування);

– тестування продуктивності програмного забезпечення, зокрема Stability testing (тестування стабільності) та Load testing (навантажувальне тестування);

– тестування інтерфейсу.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Димне тестування виконується вручну. Навантажувальне тестування проводиться за допомогою інструментарію JMeter.

Працездатність веб-застосування перевіряється шляхом:

– динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати;

– динамічного ручного тестування на відповідність функціональним вимогам;

– статичного тестування коду;

– тестування веб-ресурсу в різних браузерах;

– тестування при максимальному навантаженні;

– тестування стабільності роботи при різних умовах;

– тестування зручності використання;

– тестування інтерфейсу.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ВЕБ-ЗАСТОСУВАННЯ ДЛЯ ОРГАНІЗАЦІЇ РЕЙТИНГОВИХ ТУРНІРІВ З ЖЕРЕБКУВАННЯМ ТА ДВОМА КОНКУРУЮЧИМИ СТОРОНАМИ

Керівництво користувача

КПІ.ІП-6316.045440.05.34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Д.О. Нечай

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____Є.П. Кошовець

Київ – 2020 року

Щоб мати змогу використовувати веб-застосування, користувач повинен відкрити його в будь-якому браузері серед наступних:

- Mozilla Firefox;
- Google Chrome;
- Safari;
- Internet Explorer;
- Opera Browser.

Ролі в системі.

В даній системі є 2 ролі: гість та організатор.

Гість може:

- шукати турніри;
- переглядати турніри;
- шукати асоціації;
- переглядати асоціації;
- авторизуватись в системі;
- зареєструватись в системі.

В свою чергу Організатор може виконувати ті самі дії, що і Гість, а також:

- створювати асоціації;
- створювати учасників/команди;
- створювати турніри;
- розпочинати турнір;
- проводити раунд турніру;
- завершити турнір.

Коли користувач починає користуватись веб застосунком, то він може лише шукати турніри, асоціації та зареєструватись або автентифікуватись в системі, адже він знаходиться у ролі Гість.

Вхід в систему.

Якщо користувач хоче увійти в роль Організатор, то йому потрібно пройти автентифікацію. Треба перейти на головну сторінку застосунку або натиснути кнопку «Вхід» в навбарі.

Рисунок 1 – Панель автентифікації користувача

Користувач повинен ввести коректні логін і пароль. При правильності вводу даних – система запам’ятовує користувача, який тепер знаходиться в ролі Організатор. При введені некоректних даних буду виведено відповідну помилку на екран, як зображено на рисунку 2.

Рисунок 2 – Панель автентифікації користувача з помилкою

Якщо ще не було зареєстровано власного акаунту, то потрібно натиснути кнопку «Зареєструватися» або натиснути кнопку «Реєстрація» в навбарі. У користувача відкриється нове вікно з формою для реєстрації, зображене на рисунку 3. Користувач попинен ввести дані у відповідні поля та натиснути

кнопку «Зареєструватися». Якщо введені дані пройшли валідацію, система виведе на екран форму автентифікацію.

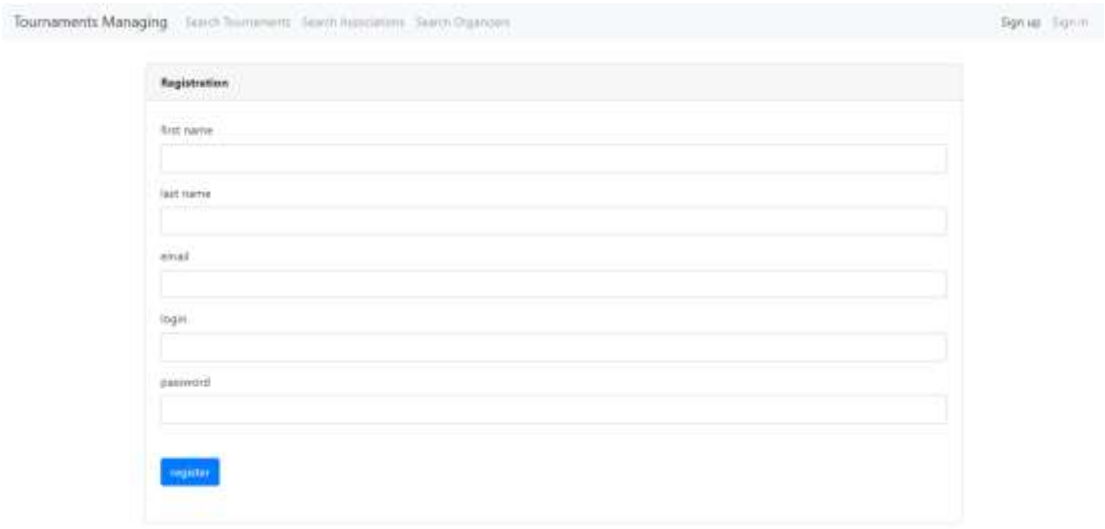


Рисунок 3 – Панель реєстрації користувача

Пошук.

Знаходячись у будь-якій ролі користувач може перейти на сторінку пошуку турнірів чи асоціацій натиснувшни кнопки «Пошук турнірів» та «Пошук асоціацій», які знаходяться в навбарі, відповідно. Приклад пошуку турнірів зображено на рисунку 4.

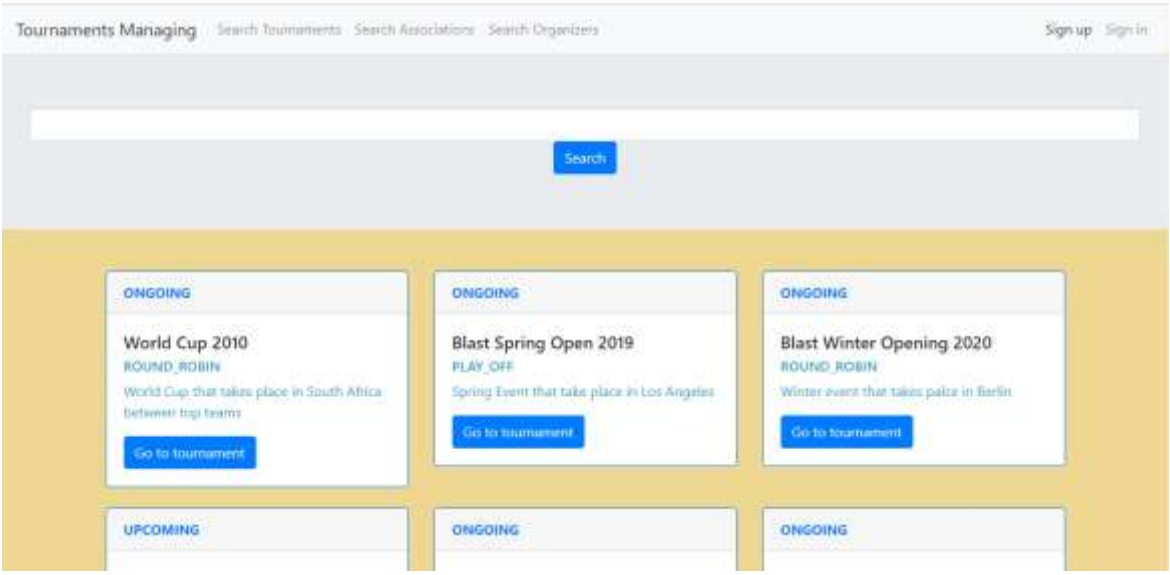


Рисунок 4 – Пошук турнірів

При першому переході на вкладку пошуку чи при введені порожніх даних в рядок пошуку, результатом будуть всі наявні турнірі чи асоціації відповідно.

Для самого пошуку нам потрібно ввести частину назви або повну її версію в рудок пошуку та натиснути кнопку «Пошук». Після цього на екрані відображаться турніри/асоціації імена яких співпадають чи містять ключове слово. За силкою можно перейти до сторінки цікавлячого користувача турніру/асоціації.

Профіль.

Організатор має свій профіль. На ньому відображена інформація про організатора та список його асоціацій. На нього можна потрапити натиснувши кнопку «Профіль» в навбарі.

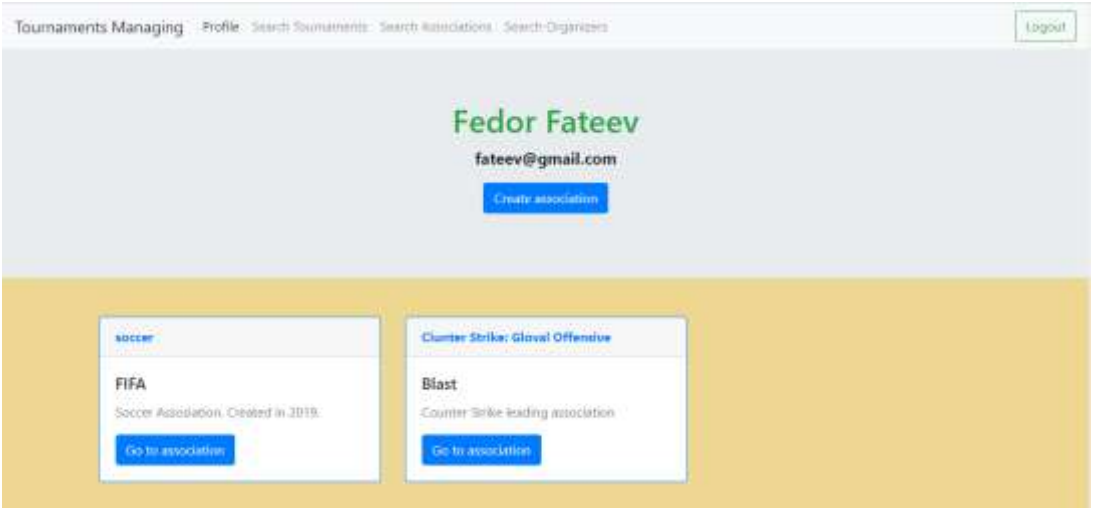


Рисунок 5 – Профіль

Асоціації предоставлені у вигляді плиток. В кожній з плиток є силка на сторінку відповідної асоціації.

Створення асоціації.

Організатор має можливість створювати свої асоціації. Щоб створити нову асоціацію треба натиснути кнопку «Створити асоціацію» на сторінці профіля організатора. Система відобразить сторінку створення асоціації, зображену на рисунку 6.

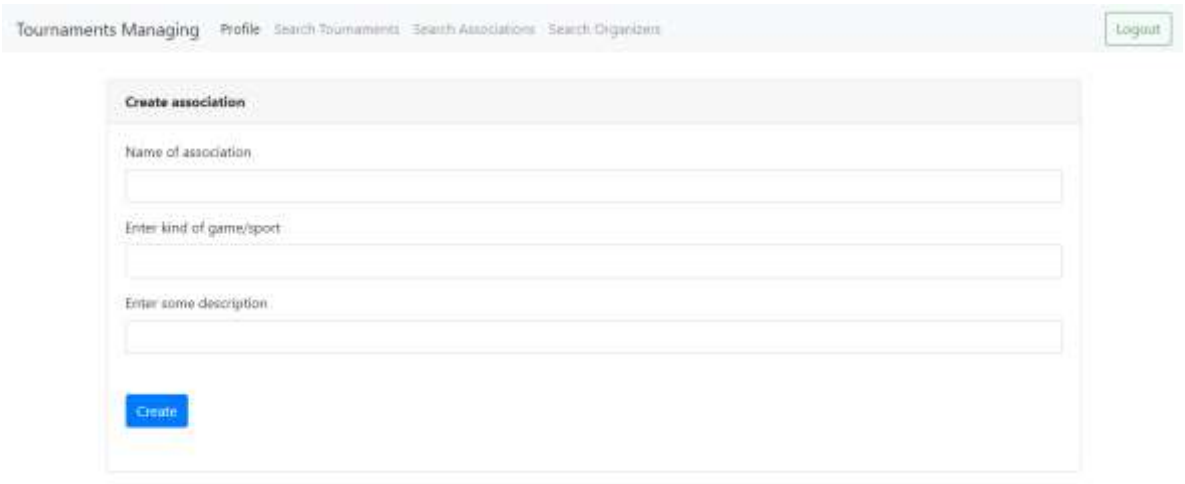


Рисунок 6 – Створення асоціації

Після введення даних в формі треба натиснути кнопку «Зберегти». Система перенаправить користувача на сторінку новоствореної асоціації.

Асоціація.

На сторінці асоціації відображаються: інформація про асоціацію, список турнірів створених в асоціації. Сторінка асоціації зображена на рисунку 7.

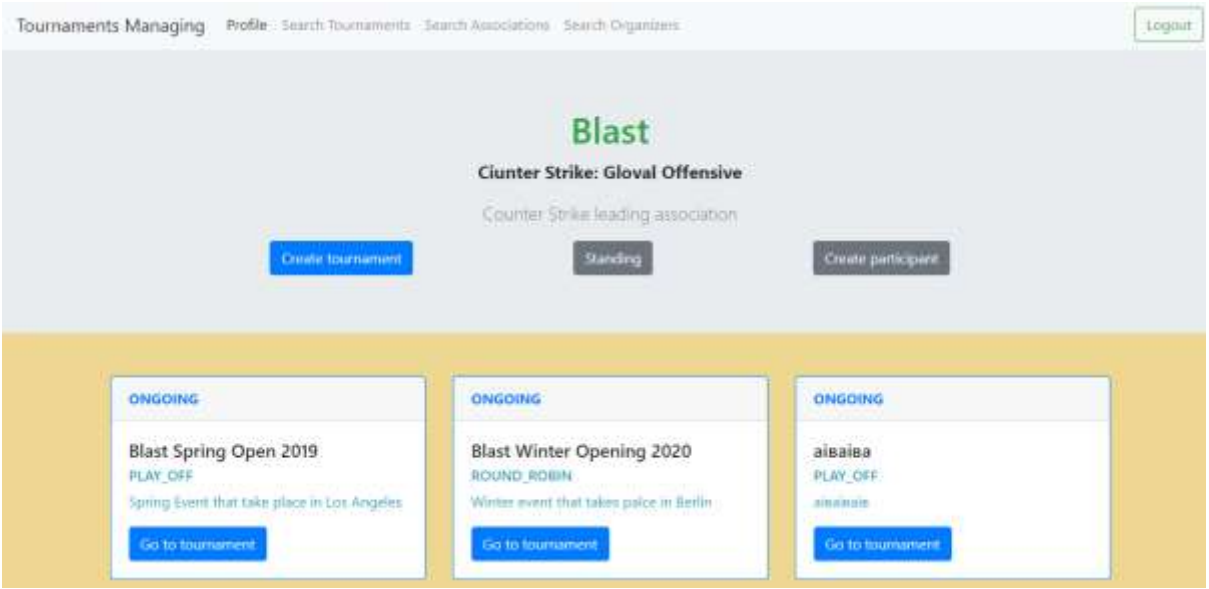


Рисунок 7 – Асоціація

Кожен користувач може переглянути положення учасників/команд в таблиці рейтингу асоціації, натиснувши кнопку «Таблиця рейтингу».

Створення учасника/команди.

Якщо організатор знаходиться на сторінці асоціації, то він може додати нових учасників/команди. Треба натиснути на кнопку «Створити

учасника/команду». Система відкриє сторінку створення учасника/команди, вона зображена на рисунку 8.

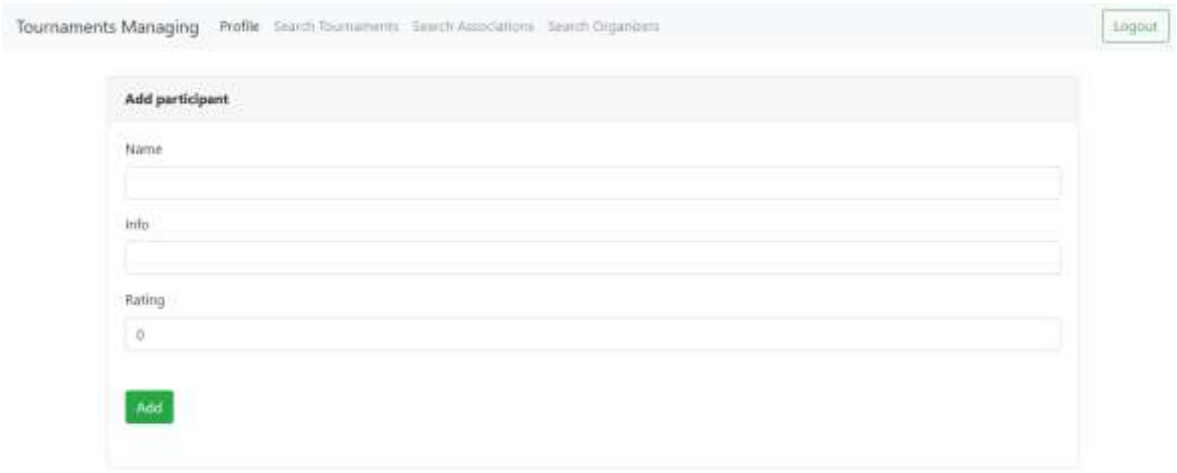


Рисунок 8 – Створення команди/учасника

Після того як користувач заповнив всі поля треба натиснути кнопку «Зберегти».Система додасть нового учасника, а користувача буде переведено на сторінку асоціації.

Створення турніру.

Якщо організатор знаходиться на сторінці асоціації, то він може створити новий турнір. Треба натиснути на кнопку «Створити турнір». Система відкриє сторінку створення турніру, вона зображена на рисунку 9.

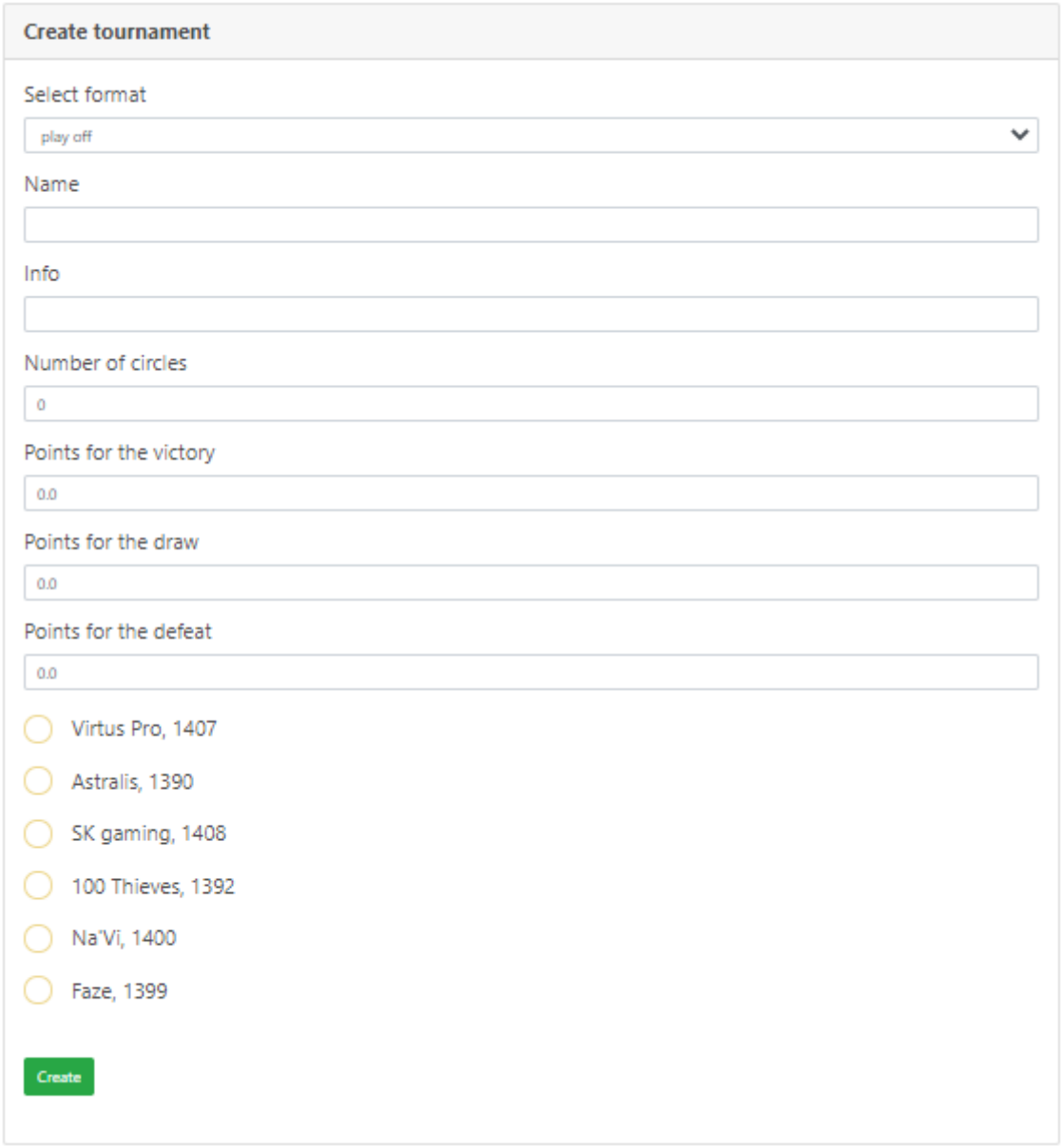


Рисунок 9 – Створення турніру

Організатор вводить назву турніру, опис, вибирає його формат, а потім вибирає учасників турніру зі списку вєе існуючих. Після цього треба натиснути кнопку «Створити». Система перейде на сторінку турніру.

Турнір.

Турнір може знаходитись у трьох станах: скоро розпочнеться, відбувається, завершено. В залежності від низ відобраєаються елементи на сторінці турніру.

Якщо турнір знаходиться в стані «Скоро розпочнеться», то сторінка виглядає як зображено на рисунку 10.

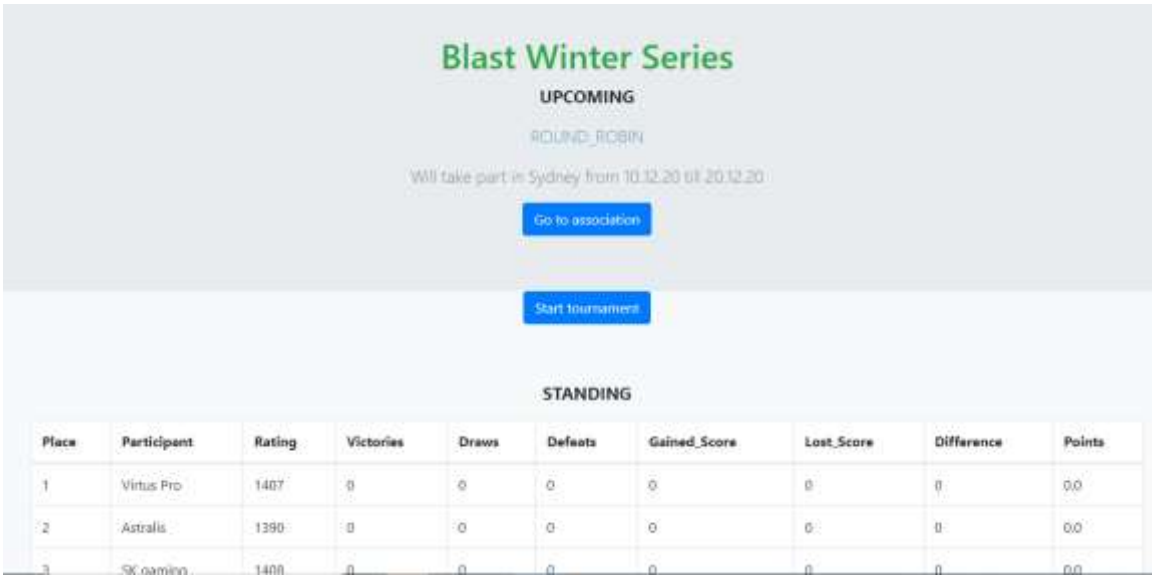


Рисунок 10 – Старт турніру

Щоб перевести турнір в стан «Відбувається» треба натиснути кнопку «Розпочати турнір».

Після цього система проводить кожен раунд жеребкування. Організатор повинен вносити результати та натискати кнопку «Закінчити раунд». Приклад вигляду таблиці результатів поточного раунду зображено на рисунку 11, становище таблиці статистики після першого ранду поточного турніру зображено на рисунку 12.

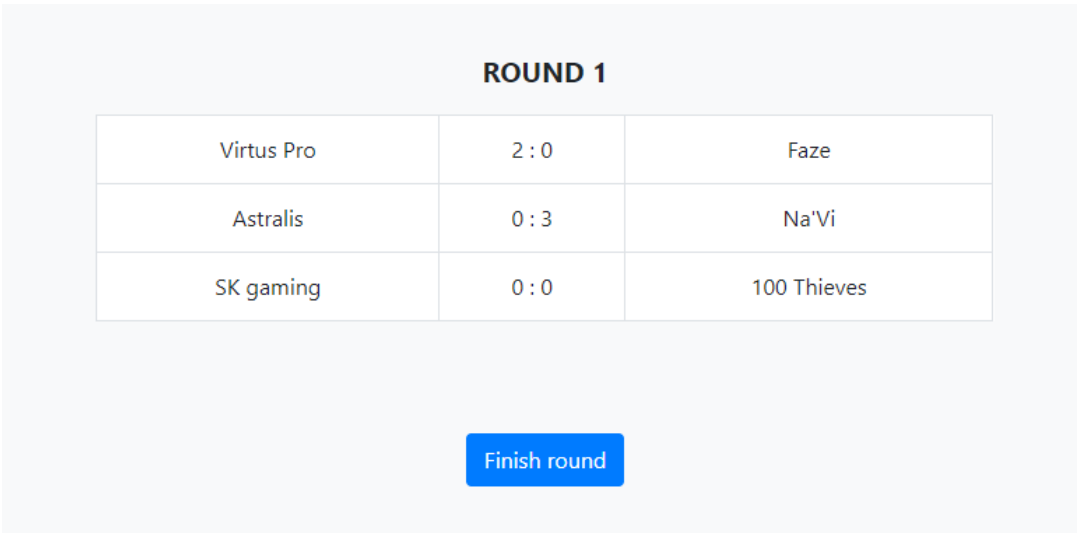


Рисунок 11 – Таблиця результатів раунду

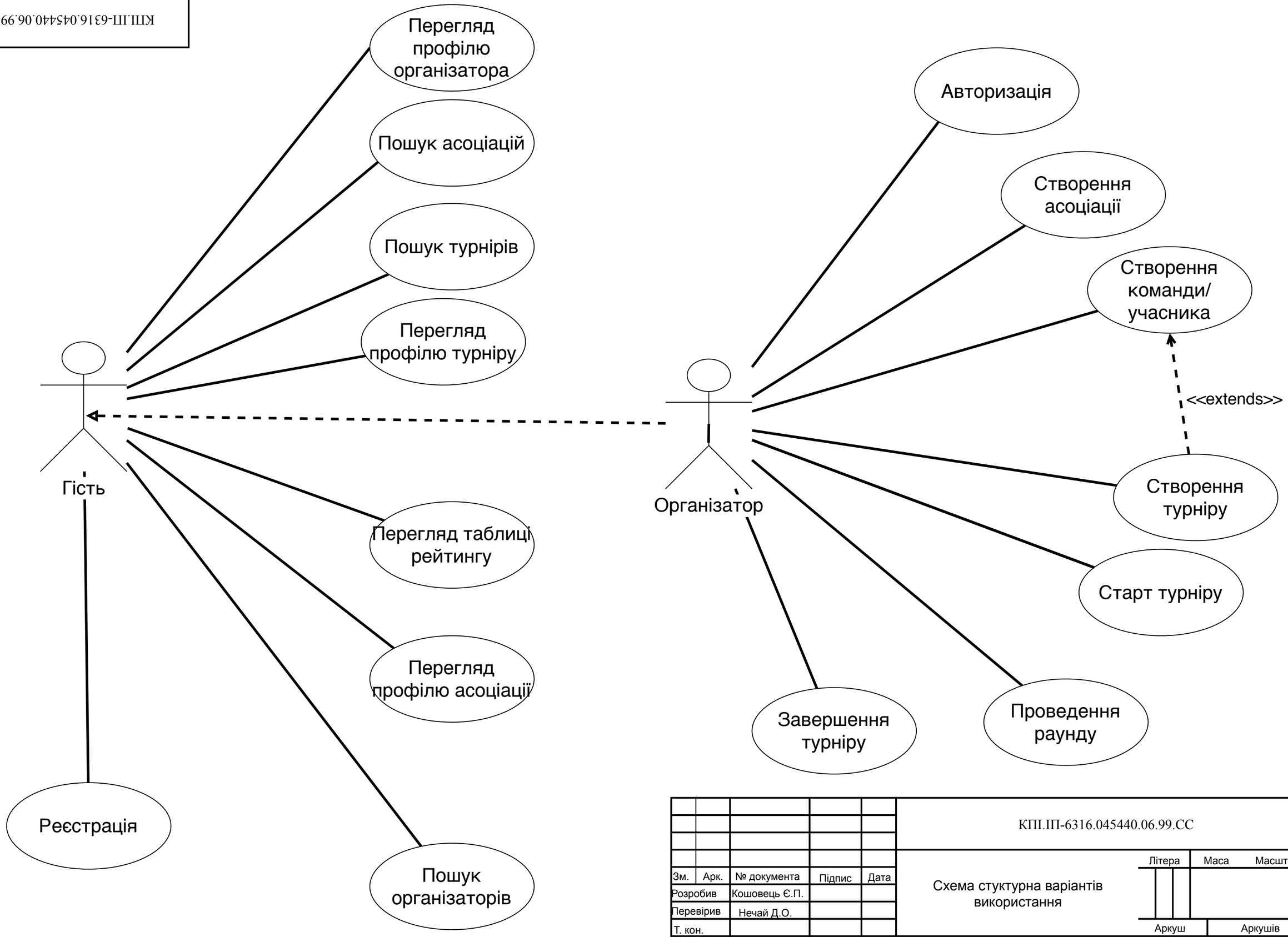
STANDING

Place	Participant	Rating	Victories	Draws	Defeats	Gained_Score	Lost_Score	Difference	Points
1	Na'Vi	1391	1	0	0	3	0	3	3.0
2	Virtus Pro	1414	1	0	0	2	0	2	3.0
3	SK gaming	1407	0	1	0	0	0	0	1.0
4	100 Thieves	1392	0	1	0	0	0	0	1.0
5	Faze	1407	0	0	1	0	2	-2	0.0
6	Astralis	1382	0	0	1	0	1	-1	0.0

Рисунок 12 – Таблиця статистики після першого раунду

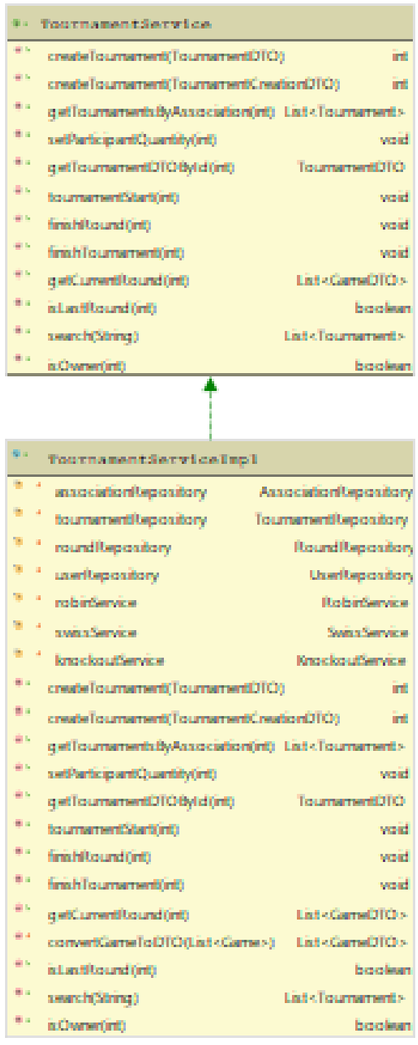
Коли відбувається останній раунд турніру, то замість кнопки «Завершити раунд» з’являється кнопка «Завершити турнір». Після її натискання турнір переходить у стан «Завершено» і відповідно завершається.

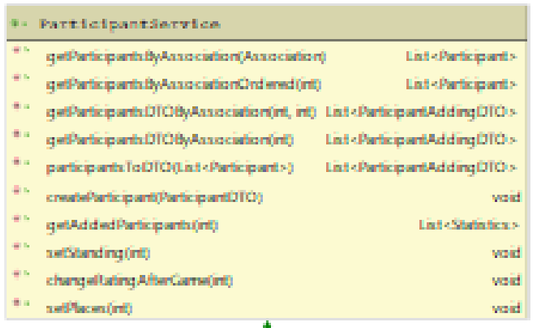
КПІ.ІП-6316.045440.06.99.СС

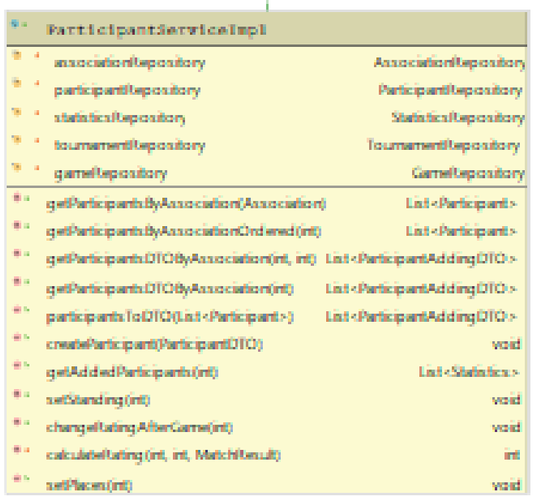


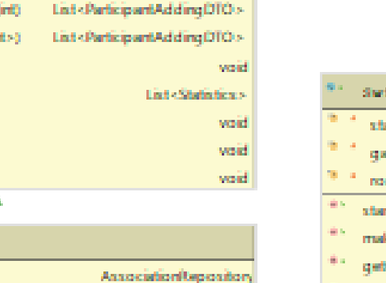
					КПІ.ІП-6316.045440.06.99.СС	
					Схема стуктурна варіантів використання	Літера: — Маса: — Масштаб: — Аркуш: — Аркушів: —
Зм.	Арк.	№ документа	Підпис	Дата		
Розробив		Кошовець Є.П.				
Перевірив		Нечай Д.О.				
Т. кон.						
Н. кон.		Ліщук К.І.			Веб-застосування організації рейтингових турнірів з жеребкуванням та двома конкуруючими сторонами	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63
Затвердив		Нечай Д.О.				

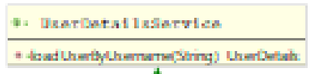
КПІ.ІП-6316.045440.07.99.СС

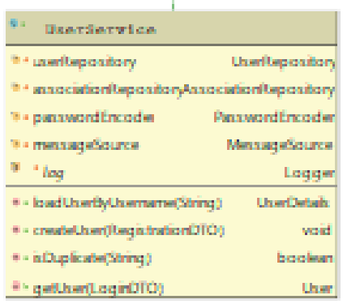


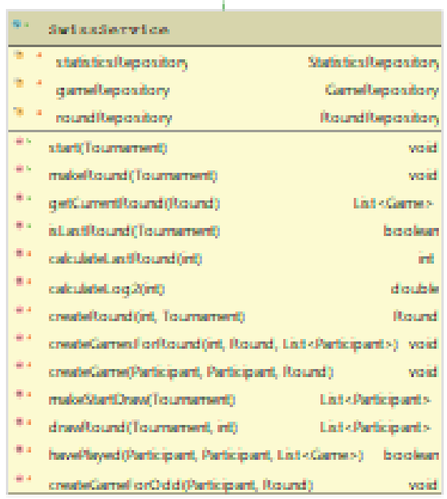


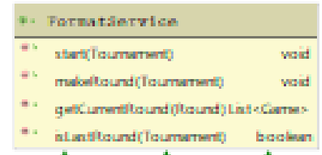


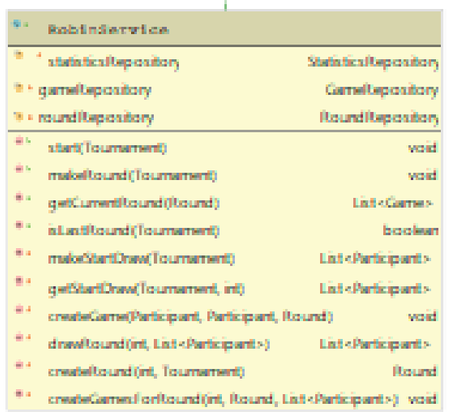


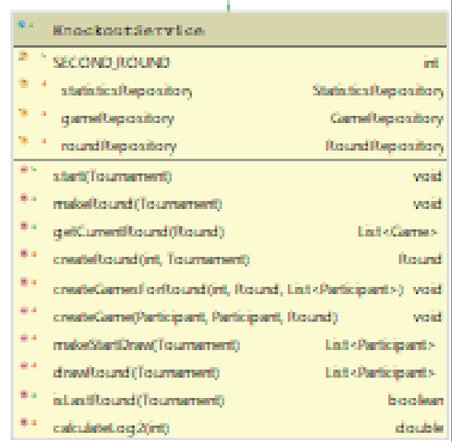


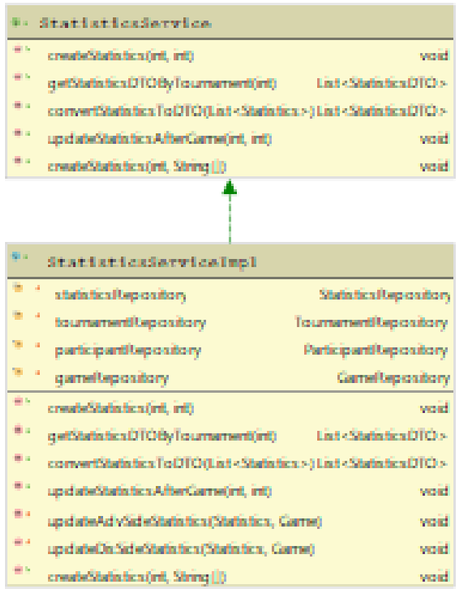


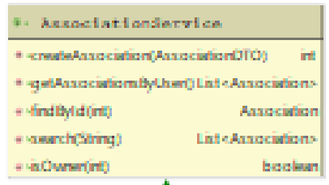


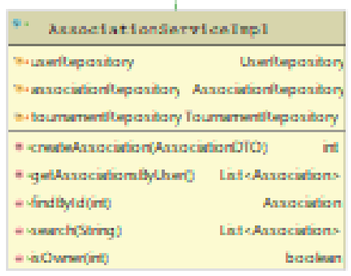




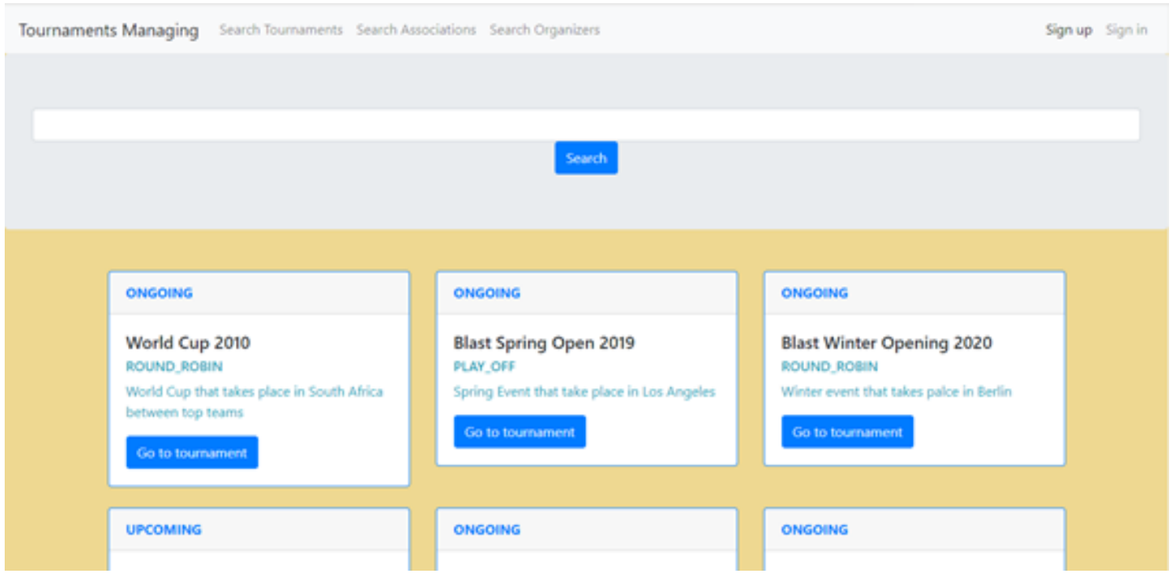


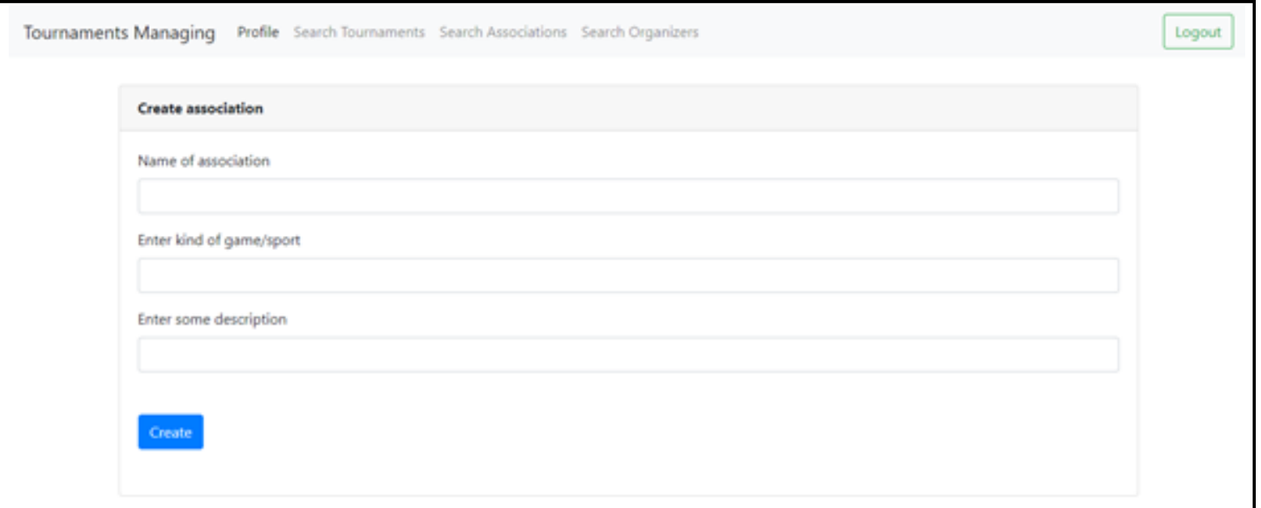


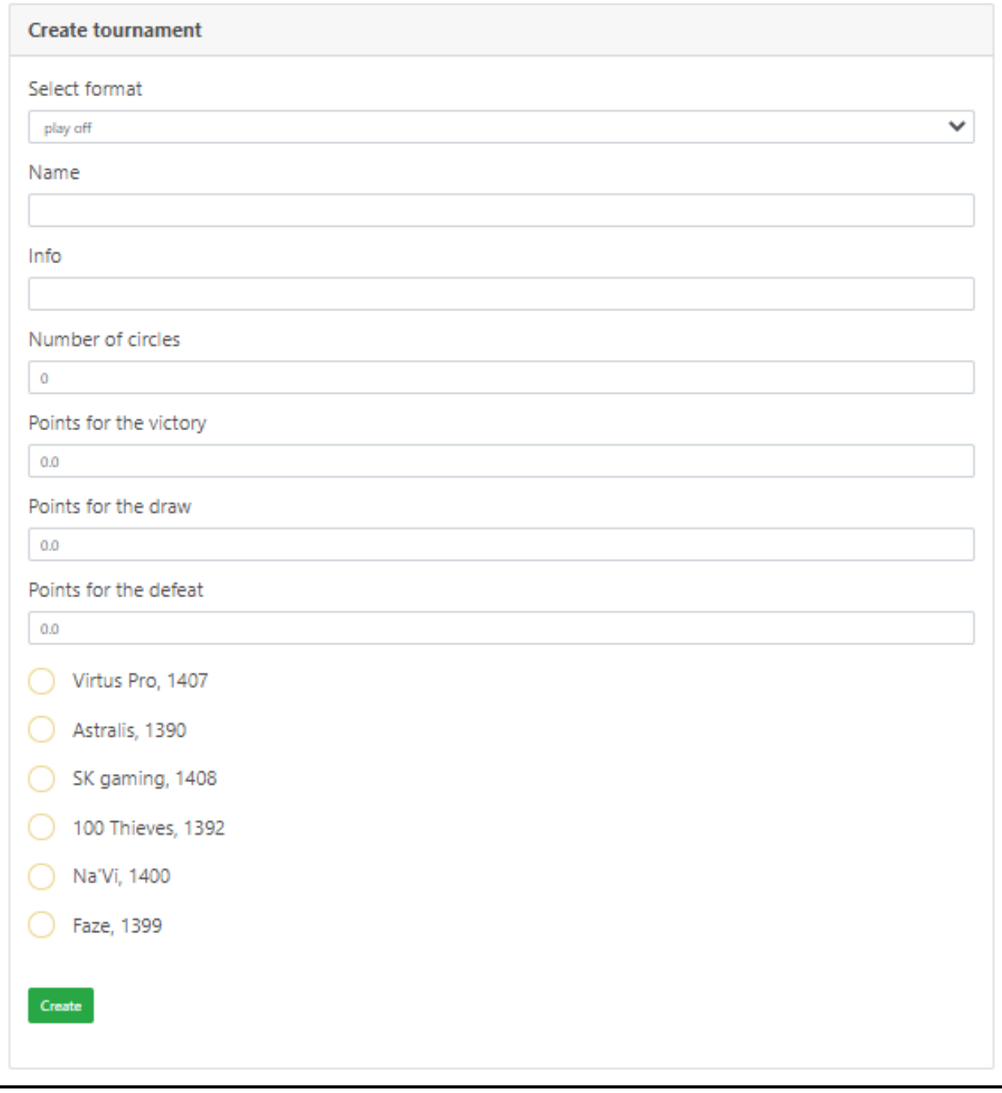


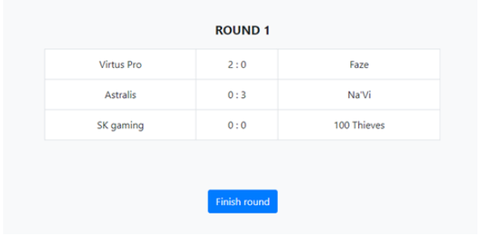


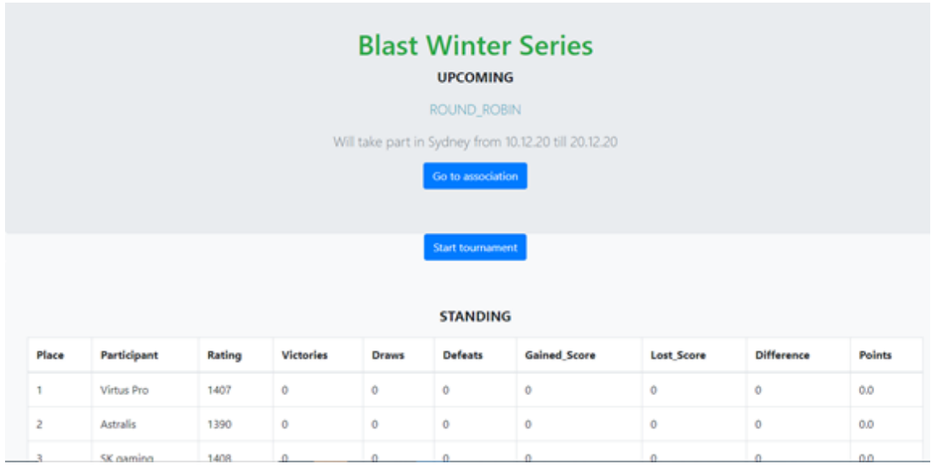
					КПІ.ІП-6316.045440.07.99.СС
Зм.	Арк.	№ документа	Підпис	Дата	Схема стуктурна класів шару сервісів
Розробив		Кошовець Є.П.			Літера: — Маса: — Масштаб: —
Перевірив		Нечай Д.О.			Аркуш — Аркушів —
Т. кон.					Веб-застосування організації рейтингових турнірів з жеребкуванням та двома конкуруючими сторонами
Н. кон.		Ліщук К.І.			КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63
Затвердив		Нечай Д.О.			

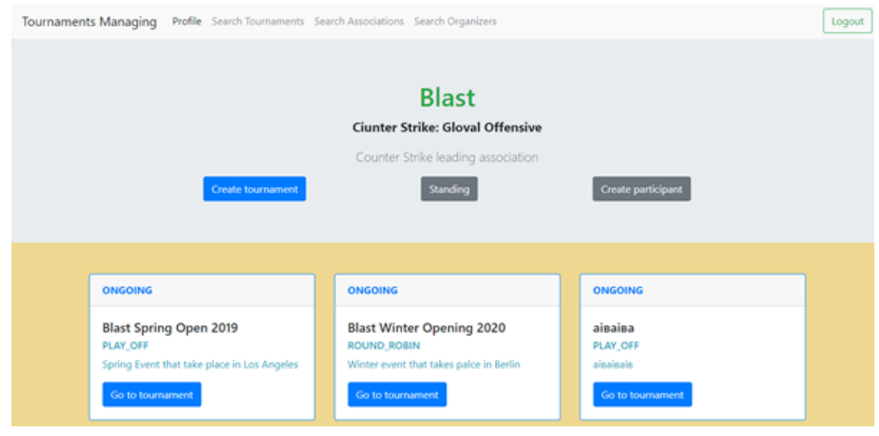












					КПІ.ІП-6316.045440.08.99.КЕ	
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм	Літера / Маса / Масштаб
Розробив		Кошовець Є.П.				
Перевірив		Нечай Д.О.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Ліщук К.І.			Веб-застосування організації рейтингових турнірів з жеребкуванням та двома конкуруючими сторонами	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63
Затвердив		Нечай Д.О.				